Supercomputing Frontiers and Innovations

2017, Vol. 4, No. 2

Scope

- Enabling technologies for high performance computing
- Future generation supercomputer architectures
- Extreme-scale concepts beyond conventional practices including exascale
- Parallel programming models, interfaces, languages, libraries, and tools
- Supercomputer applications and algorithms
- Distributed operating systems, kernels, supervisors, and virtualization for highly scalable computing
- Scalable runtime systems software
- Methods and means of supercomputer system management, administration, and monitoring
- Mass storage systems, protocols, and allocation
- Energy and power minimization for very large deployed computers
- Resilience, reliability, and fault tolerance for future generation highly parallel computing systems
- Parallel performance and correctness debugging
- Scientific visualization for massive data and computing both external and in situ
- Education in high performance computing and computational science

Editorial Board

Contents

Using High Performance Computing to Create and Freely Distribute the South Asian Genomic Database, Necessary for Precision Medicine in this Population
A.H. Shah, J.D. Picker, S.S. Jamuar 4

An Application of GPU Acceleration in CFD Simulation for Insect Flight
Y. Yao, K.-S. Yeo 13

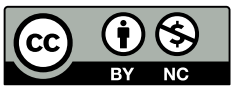
Simultac Fonton: A Fine-Grain Architecture for Extreme Performance beyond Moore's Law
M. Brodowicz, T. Sterling 27

The Simultaneous Transmit And Receive (STAR) Message Protocol
E. Jennings 38

Core Module Optimizing PDE Sparse Matrix Models with HPCG Example
E. Jennings 54

Beating Floating Point at its Own Game: Posit Arithmetic
J.L. Gustafson, I. Yonemoto 71

InfiniCortex - From Proof-of-concept to Production
G. Noaje, A. Davis, J. Low, L. Seng, T.G. Lian, L.P. Orlowski, D. Chien, L.S. Wu, T.T. Wee, Y. Poppe, B.H.K. Kenneth, A. Howard, D. Southwell, J. Gunthorpe, M.T. Michalewicz 87



DOI: 10.14529/jsfi170201

Using High Performance Computing to Create and Freely Distribute the South Asian Genomic Database, Necessary for Precision Medicine in this Population

Asmi H. Shah[1], *Jonathan D. Picker*[1], *Saumya S. Jamuar*[1]

Precision medicine is an emerging approach for disease treatment and prevention that takes into account individual variability in genes, environment, and lifestyle for each person. Efforts to implement precision medicine have gained traction in recent years due to significantly increased understanding of the role of genetic variations in human disease over the past decade. However, delivery of precision medicine requires robust population specific reference genome datasets for full appreciation of existing natural variation. A large majority of publicly available genomic databases are primarily derived from Caucasian populations and do not fully address the diversity of Asian populations. In an effort to address this problem, we have aggregated and built a genomic database, ggcINDIA, specifically for South Asian populations. In collaboration with Global Alliance for Genomics and Health (GA4GH), we have made this database publicly available to the community through the GA4GH's Beacon project. ggcINDIA represents the first Beacon for South Asian populations. As more data are generated and aggregated, the ggcINDIA beacon will provide the precise genomic data that is critical to the delivery of precision medicine within South Asia.

Keywords: Precision Medicine, ggcINDIA, Beacon Network, South Asian Genome, Genome Data Sharing.

Introduction

Next generation sequencing and constant advances in the high throughput technologies as well as lab automation have made it possible to explore the vast variation that exists within the human genome [8, 14]. For example, variations in genes related to drug metabolism (also known as pharmacogenomics) such as *CYP2C19, NAT2,* etc. affect the individual's response to drug treatment. Similarly, presence of specific pathogenic variants in certain cancers allow the use of targeted therapeutics (also known as precision oncology). For example, treatment of melanoma with a somatic *V600E* variant in the *BRAF* gene, specifically includes the use of selective *BRAF* inhibitors such as *vemurafenib*. Selective inhibition of *BRAF* results in a relative reduction of 63% in risk of death and 74% in risk of tumor progression [2]. These success stories have accelerated the move towards precision medicine, a disruptive model of healthcare delivery, where treatment is tailored to the individual's characteristics, in most cases, the genetic or molecular information.

For successful delivery of precision medicine, it is imperative to understand the genomic variations, and their consequences, for different populations. Studies such as the 1000 Genome project have demonstrated that these genetic variations are dependent on the ancestry and ethnicity [6, 21]. They are responsible for the phenotypic diversity within the diverse populations, such as facial appearance, but more importantly, for differences in disease susceptibility and therapeutic response. A major limitation of previous genomic studies was a focus on Caucasian populations. More recent efforts have have begun to individuals from a more diverse non-Caucasian background, which has led to an increase in representation of non-European individuals in the NHGRI-EBI GWAS from 4% in 2009 to 19% in 2016 [18]. However, the vivid diversity of South Asian population [20], that is not accurately represented even with the increased representation

[1]Global Gene Corporation Pte Ltd, Singapore, Singapore

of non-European individuals in publicly available genomic databases [13, 18]. The concern with this bias is that it can result in misinterpretations and misdiagnoses [15]. In a recent analysis by the Exome Aggregation Consortium group at the Broad Institute, only 9 out of 192 variants, previously called as pathogenic, were truly pathogenic, while over 160 variants were population specific polymorphisms, and hence, likely benign [13, 23]. Furthermore, on comparing the standard datasets to bushmen in the KB1 African genome analysis, it was found that there was an increased frequency of sequence variation between them, and over 47% of variants identified were novel, affecting over 7700 genes; indicating the scale of population diversity [22].

To fill in the gap of South Asian genome knowledge, the goal of this study is to build genome database for South Asian populations and to make this database accessible to researchers and at large.

1. Methods

1.1. Aggregating Genomic Data of South Asian populations

Through separate projects, we identified individuals of South Asian ancestry, and performed genomic sequencing (either whole exome or whole genome) on these individuals. In addition, we aggregated available genomic data, including some available publicly through Creative Commons Attribution License [4]. All of the individuals included in this study had all of their four grandparents born on the Indian subcontinent.

The DNA sequence of each individual was collected in the standard FASTQ files, which store the DNA sequence as well as its corresponding quality scores.

1.2. Genome Analysis: Alignment, Variant Calling, and Variant Annotation

A customized bioinformatic analysis pipeline was set up to analyze the genome sequences and make the South Asian variant datasets available to the public (fig. 1).

The raw genomics data in FASTQ format were processed using the Sentieon DNAseq pipeline version 201611 [9]. Sentieon DNAseq is a proprietary reimplementation of Broad Institutes best practices pipeline for DNAseq [3] with an approximately 10x improvement in runtime. Performance improvements were achieved through use of Sentieon's proprietary improved algorithms and better resource management.

DNA sequences in FASTQ files were aligned to the known and publicly available reference genome GRCh37 using Sentieon BWA (sequence alignment tool) and the resulting alignments were sorted by genomic coordinates and converted to BAM format (binary format of sequence data) using the Sentieon UTIL binary. The quality of the aligned sequence data - including mean base quality for each flowcell cycle, the base quality score distribution, GC bias metrics, alignment metrics, and insert size metrics were calculated for each sample using the Sentieon driver. Duplicates in the sequences were removed, reads were realigned around indels (insertions and deletions in the sequence) identified by the 1000 Genomes project [6] or Mills et al. [17], and base quality scores were recalibrated. (The variation in the DNA sequence that occurs at a specific position in the genome is called a variant.) Variants were called for each sample independently using the Sentieon DNAseq Haplotyper and variants were output as genomic Variant Call Format (gVCF) files. Joint genotyping was performed on all gVCF files using the

Sentieon DNAseq GVCFtyper. This step creates a common VCF file having the information from all the individuals' sequences.

Variant Effect Predictor (VEP) version 86 from Ensembl was used to annotate the VCF for further analysis. VEP determines the effect of variants (including single nucleotide polymorphisms (SNPs), insertions and deletions) on genes, transcripts, and protein sequence, as well as regulatory regions [16].

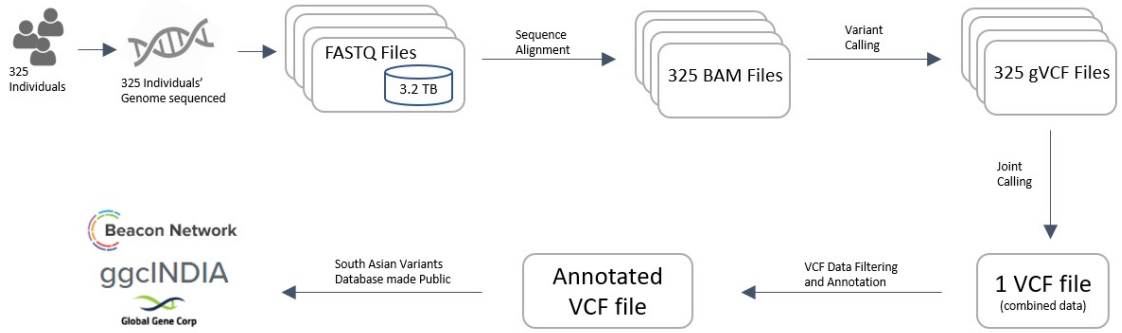


Figure 1. Genome Sequence Analysis of 325 individuals DNA sequences. From raw data genome files to making the Variants of South Asian populations public on Beacon Network search engine

1.3. Computation and Resources

High performance computing infrastructure provided through the National Super Computing Centre, Singapore was used to perform memory, resource, and compute intensive operations. PBS Pro was used to manage the workload and use the HPC resources efficiently. 2 units of 24 CPUs, 96GB of memory, 1 GPU, and 12 threads of 2 MPI processes were used. The total size of the raw data was 3183.3 GB.

Sentieon's DNAseq pipeline as well as VEP were deployed onto the compute nodes on the NSCC server infrastructure. Processing one individual's FASTQ files to a gVCF file took about 6 hours on a single 32 core server. 64GB of memory is recommended to process such a sample. The computation time can be reduced to under an hour using distributed computing processes on multiple parallel servers. In our case, with the above mentioned resource configuration, the job of processing 325 individuals' genome datasets was completed in 168 hours.

2. Results

2.1. Demographics

We aggregated genomic data from 325 individuals of South Asian ancestry (tab. 1). Out of the 234 individuals where data on geographical distribution within India was available, 67.2% were from North India, 15.9% were from South India, 14.7% were from West India and 2.2% were from East India. There were 291 (89.5%) males. The age range was 31 to 81 years (median 48 years).

Table 1. Distribution of 325 Individuals by their Country of Birth. For Each of them, all 4 grandparents were native to the Indian Subcontinent

COUNTRY OF BIRTH	PROPORTION(%)
India	72.0% (234)
Pakistan	10.5% (34)
Sri Lanka	5.2% (17)
Bangladesh	1.2% (4)
Afghanistan	0.6% (2)
Other	10.5% (34)

2.2. Genomic Data

All individuals underwent genomic sequencing as per standard protocols. 178 underwent whole genome sequencing and 147 underwent whole exome sequencing. All sequencing was performed on the Illumina platform.

2.3. Genomic Variant Calling and Annotation

Variants in one's genome are defined as the differences in an individual's genome when compared to a reference genome. These variants account for the differences among individuals and tend to cluster based on ancestry [7]. While some of these variations may directly alter the structure or function of the protein they code (also known as protein altering variants), a significant majority of these variants occur in the non protein coding regions, and the significance of these variants has not been well elucidated. Some of the variants could have associations with human diseases or complex traits.

In our cohort, we detected 19,643,311 variants, which were then annotated using VEP. The majority of the variants (81.6%) were single nucleotide variants (SNV) (fig. 2) (replacement of a single nucleotide in the sequence). The rest of the 18.4% variants are indels and sequence alteration - meaning there were insertions or deletions of nucleotide(s) from the sequence. Only 1.1% of these SNV variants were coding (coding region is the part which translates into proteins), while 47.1% were intronic and 39.7% were intergenic (fig. 3). Among the coding variants, 54.0% were missense variants, 42.0% were synonymous variants, 1.5% were frameshift variants and 1.0% affected the termination codon (fig. 4). Among the missense coding variants, 59.8% were predicted to be benign by Polyphen-2 [1], while 33.7% were predicted to be either possibly or probably damaging (suppl. fig. 1). Distribution of variants across chromosomes is demonstrated in suppl. fig. 2 to 27. In each of the figures, the X-axis is a position along the particular chromosome and the Y-axis is a number of variants at the given location. This distribution, in turn, does show the areas of increased variation.

2.4. ggcINDIA Beacon

Beacon Network by Global Alliance for Genomics and Health (GA4GH) is a global search engine for genetic mutations [10]. Each collaborator's genomic datasets in the form of VCF files are uploaded and is called lighting a 'Beacon'. It enables global discovery of genetic mutations,

Figure 2. Variant classification shown for the total of 19,643,311 variants detected among the 325 individuals. SNV= single nucleotide variant

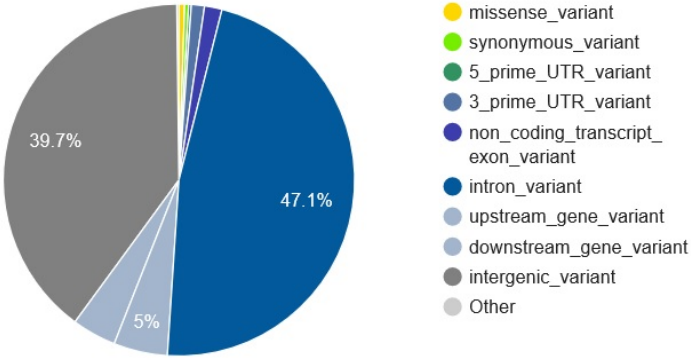


Figure 3. Variant classification depending on the consequences, shows the most severe ones. Only 1.1% are coding sequence variants

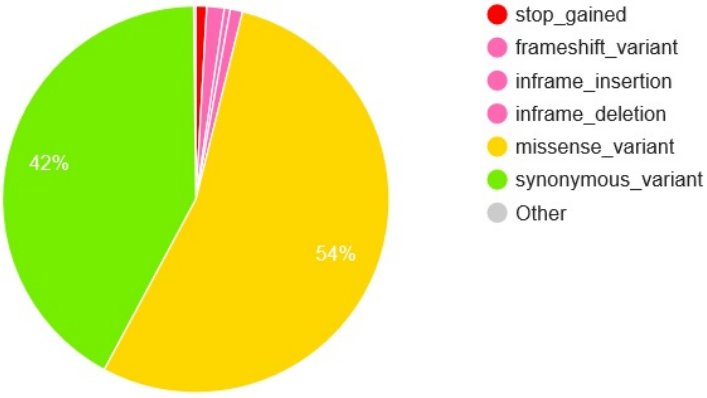


Figure 4. Coding consequences, variant classification of the 1.1% coding variants

federated across a large and growing network of shared genetic datasets. To join the effort and to contribute to the community, we chose Beacon Network as the platform to make our South Asian populations' variant database public.

In collaboration with GA4GH, we have published the South Asian genomic variant database, the first 'beacon' of its kind for the South Asian population called ggcINDIA (fig. 5). This beacon is the 69th beacon in the network. The beacon is a freely available resource and allows researchers and the public to query the presence or absence of a given variant detected in their own discovery cohort, and allows for filtering of variants for rarity. Once you access ggcINDIA, you can filter out the variants specifically within the South Asian population.

Over time, we foresee generating and aggregating more genome sequences of individuals from various cohorts of South Asian ethnicity into ggcINDIA beacon.

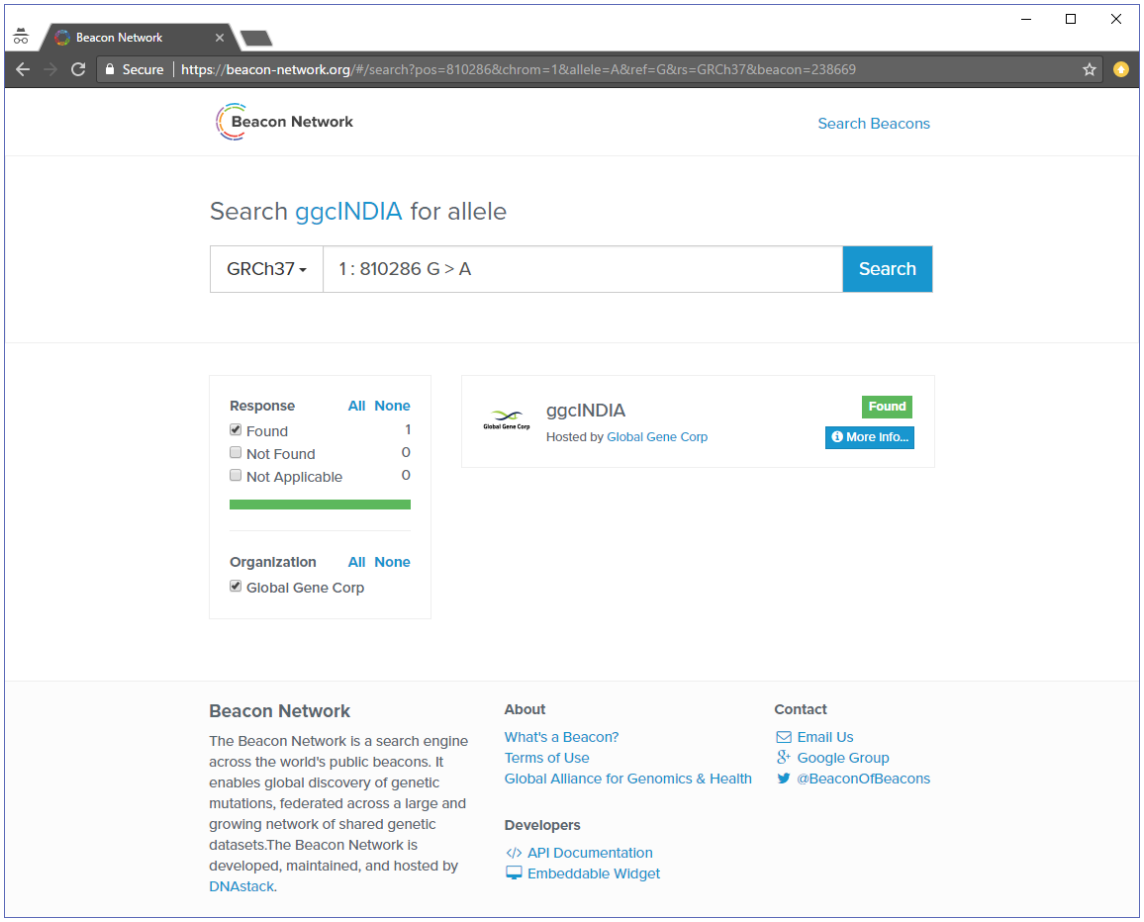


Figure 5. ggcINDIA on the Beacon Network. This interface allows researchers to know if a particular searched variant is present or absent in the population. Here, the searched variant was found in ggcINDIA.

3. Discussion

The correlation of genetic information with drug interactions as well as phenotypic and pathogenic traits has proven that healthcare can be improved by personalizing to ones characteristics and treatments [5]. Precision medicine is changing the dynamics of how healthcare is delivered. However, for precision medicine to have maximum impact, the genomics of diverse population cohorts must be known. The majority of known genetic knowledge is derived from Caucasian populations [18]. The relative frequency of alleles important for pharmacogenomics varies by population, meaning that certain drugs or drug groups will be less effective or even hazardous in some populations, *e.g.*, the risk for toxic epidermal necrolysis with the antiepileptic drug carbamazepine in East Asians [24], and more effective and safer in other such as statin use in Iranians with a specific *KIF6* variant [11].

ggcINDIA is an initiative that takes up the challenge to recruit the under-represented populations and add their genomic information to correct the known racial bias of currently available genomic knowledge. This study supports the fact that scientific data needs to be shared and made publicly available within the scientific community as well as the public [12, 19]. ggcINDIA

is part of global data sharing movement lead by GA4GH [19] and their flagship program of Beacon Network. Such initiatives will only widen the scope of the reference genome and take the necessary and obvious diversity into account.

ggcINDIA made its start with 325 individuals' genomic data. Our aim is to continue to grow and add data from more individuals to create a high fidelity South Asian reference genome. Thus, moving forwards, we invite other collaborators to come and share their genomic datasets for South Asian population and contribute in increasing the fidelity of the database. This process will provide more accurate genomic data that is critical to delivery of precision medicine within South Asia.

Acknowledgements

We would like to thank Sentieon Inc., USA for providing us access to the Sentieon DNAseq pipeline and Donald Freed from Sentieon Inc., USA for his assistance with the manuscript. We are also grateful for the assistance from the Singapore National Supercomputing Centre (NSCC), Global Alliance for Genomics and Health, European Bioinformatics Institute (EBI), Sanger Institute and DNAStack. Some of the data used in this study was made available through the Creative Commons Attribution License (© 2014 Chambers et al.).

References

1. Adzhubei, I., Jordan, D.M., Sunyaev, S.R.: Predicting functional effect of human missense mutations using polyphen-2. Current protocols in human genetics pp. 7–20 (2013), DOI:10.1002/0471142905.hg0720s76

2. Ascierto, P.A., Kirkwood, J.M., Grob, J.J., Simeone, E., Grimaldi, A.M., Maio, M., Palmieri, G., Testori, A., Marincola, F.M., Mozzillo, N.: The role of braf v600 mutation in melanoma. Journal of translational medicine 10(1), 85 (2012), DOI:10.1186/1479-5876-10-85

3. Auwera, G.A., Carneiro, M.O., Hartl, C., Poplin, R., del Angel, G., Levy-Moonshine, A., Jordan, T., Shakir, K., Roazen, D., Thibault, J., et al.: From fastq data to high-confidence variant calls: the genome analysis toolkit best practices pipeline. Current protocols in bioinformatics pp. 11–10 (2013), DOI:10.1002/0471250953.bi1110s43

4. Chambers, J.C., Abbott, J., Zhang, W., Turro, E., Scott, W.R., Tan, S.T., Afzal, U., Afaq, S., Loh, M., Lehne, B., et al.: The south asian genome. PLoS One 9(8), e102645 (2014), DOI:10.1371/journal.pone.0102645

5. Collins, F.S., Varmus, H.: A new initiative on precision medicine. New England Journal of Medicine 372(9), 793–795 (2015), DOI:10.1056/NEJMp1500523

6. Consortium, .G.P., et al.: A global reference for human genetic variation. Nature 526(7571), 68–74 (2015), DOI:10.1038/nature15393

[2] https://creativecommons.org/licenses/by-nc/3.0/

7. Cotton, R., Horaitis, O.: Human genome variation society. eLS (2006), DOI:10.1038/npg.els.0005964

8. DePristo, M.A., Banks, E., Poplin, R., Garimella, K.V., Maguire, J.R., Hartl, C., Philippakis, A.A., Del Angel, G., Rivas, M.A., Hanna, M., et al.: A framework for variation discovery and genotyping using next-generation dna sequencing data. Nature genetics 43(5), 491–498 (2011), DOI:10.1038/ng.806

9. Freed, D.N., Aldana, R., Weber, J.A., Edwards, J.S.: The sentieon genomics tools-a fast and accurate solution to variant calling from next-generation sequence data. bioRxiv p. 115717 (2017), DOI:10.1101/115717

10. GlobalAllianceForGenomics&Health: Beacon network. `https://beacon-network.org`, accessed: 2017-05-09

11. Hamidizadeh, L., Abadi, B., Hosseini, R.H., Baigi, B., Ali, M., Dastsooz, H., Nejhad, A.K., Fardaei, M.: Impact of kif6 polymorphism rs20455 on coronary heart disease risk and effectiveness of statin therapy in 100 patients from southern iran. Archives of Iranian Medicine (AIM) 18(10) (2015)

12. Kosseim, P., Dove, E.S., Baggaley, C., Meslin, E.M., Cate, F.H., Kaye, J., Harris, J.R., Knoppers, B.M.: Building a data sharing model for global genomic research. Genome biology 15(8), 430 (2014), DOI:10.1186/s13059-014-0430-2

13. Lek, M., Karczewski, K.J., Minikel, E.V., Samocha, K.E., Banks, E., Fennell, T., ODonnell-Luria, A.H., Ware, J.S., Hill, A.J., Cummings, B.B., et al.: Analysis of protein-coding genetic variation in 60,706 humans. Nature 536(7616), 285–291 (2016), DOI:10.1038/nature19057

14. Levy, S.E., Myers, R.M.: Advancements in next-generation sequencing. Annual review of genomics and human genetics 17, 95–115 (2016), DOI:10.1146/annurev-genom-083115-022413

15. Manrai, A.K., Funke, B.H., Rehm, H.L., Olesen, M.S., Maron, B.A., Szolovits, P., Margulies, D.M., Loscalzo, J., Kohane, I.S.: Genetic misdiagnoses and the potential for health disparities. New England Journal of Medicine 375(7), 655–665 (2016), DOI:10.1056/NEJMsa1507092

16. McLaren, W., Gil, L., Hunt, S.E., Riat, H.S., Ritchie, G.R., Thormann, A., Flicek, P., Cunningham, F.: The ensembl variant effect predictor. Genome biology 17(1), 122 (2016), DOI:10.1186/s13059-016-0974-4

17. Mills, R.E., Luttig, C.T., Larkins, C.E., Beauchamp, A., Tsui, C., Pittard, W.S., Devine, S.E.: An initial map of insertion and deletion (indel) variation in the human genome. Genome research 16(9), 1182–1190 (2006), DOI:10.1101/gr.4565806

18. Popejoy, A.B., Fullerton, S.M.: Genomics is failing on diversity. Nature 538(7624), 161 (2016), DOI:10.1038/538161a

19. Rahimzadeh, V., Dyke, S.O., Knoppers, B.M.: An international framework for data sharing: Moving forward with the global alliance for genomics and health. Biopreservation and biobanking 14(3), 256–259 (2016), DOI:10.1089/bio.2016.0005

20. Reich, D., Thangaraj, K., Patterson, N., Price, A.L., Singh, L.: Reconstructing indian population history. Nature 461(7263), 489–494 (2009), DOI:10.1038/nature08365

21. Rotimi, C.N., Jorde, L.B.: Ancestry and disease in the age of genomic medicine. New England Journal of Medicine 363(16), 1551–1558 (2010), DOI:10.1056/NEJMra0911564

22. Schuster, S.C., Miller, W., Ratan, A., Tomsho, L.P., Giardine, B., Kasson, L.R., Harris, R.S., Petersen, D.C., Zhao, F., Qi, J., et al.: Complete khoisan and bantu genomes from southern africa. Nature 463(7283), 943–947 (2010), DOI:10.1038/nature08795

23. Song, W., Gardner, S.A., Hovhannisyan, H., Natalizio, A., Weymouth, K.S., Chen, W., Thibodeau, I., Bogdanova, E., Letovsky, S., Willis, A., et al.: Exploring the landscape of pathogenic genetic variation in the exac population database: insights of relevance to variant classification. Genetics in Medicine 18(8), 850–854 (2015), DOI:10.1038/gim.2015.180

24. Tangamornsuksan, W., Chaiyakunapruk, N., Somkrua, R., Lohitnavy, M., Tassaneeyakul, W.: Relationship between the hla-b* 1502 allele and carbamazepine-induced stevens-johnson syndrome and toxic epidermal necrolysis: a systematic review and meta-analysis. JAMA dermatology 149(9), 1025–1032 (2013), DOI:10.1001/jamadermatol.2013.4114

DOI: 10.14529/jsfi170202

An Application of GPU Acceleration in CFD Simulation for Insect Flight

Yang Yao[1], ***Khoon-Seng Yeo***[1]

The mobility and maneuverability of winged insects have been attracting attention, but the knowledge on the behavior of free-flying insects is still far from complete. This paper presents a computational study on the aerodynamics and kinematics of a free-flying model fruit-fly. An existing integrative computational fluid dynamics (CFD) framework was further developed using CUDA technology and adapted for the free flight simulation on heterogeneous clusters. The application of general-purpose computing on graphics processing units (GPGPU) significantly accelerated the insect flight simulation and made it less computational expensive to find out the steady state of the flight using CFD approach. A variety of free flight scenarios has been simulated using the present numerical approach, including hovering, fast rectilinear flight, and complex maneuvers. The vortical flow surrounding the model fly in steady flight was visualized and analyzed. The present results showed good consistency with previous studies.

Keywords: insect flight, free flight, general-purpose computing on graphics processing units (GPGPU), computational fluid dynamics (CFD), flapping-wing aerodynamics.

Introduction

Winged insects were the first animals to evolve flight locomotion. It is common to find them hovering, flying sideways, landing up-side down, and executing rapid changes in flight speed and direction. The capability of flight significantly contributes to insect diversity and abundance, as it has permitted access to various ecological resources and rapid escape from predators. The unparalleled mobility and maneuverability of winged insects have long captured the interest of researchers in the area of biomechanics and aerodynamics. It is believed that, by explaining the biomechanics of insect flight, scholars can yield insight to morphological evolution of insects and their ecological roles.

In the past decade, several unsteady aerodynamic phenomena have been discovered to be responsible for high agility and lift enhancement of flapping wing insect [12]. With the development of CFD methods, and the continuous improvement of computing technology, it has become feasible to study free flying insects and to visualize and analyze the complex unsteady aerodynamics using numerical approaches [9, 13]. The use of CFD provides a convenient approach to model various natural flapping wing flyers, since the geometries of the flyers and the environment the flyers encounter can be easily altered in a CFD model once it has been developed. Sun & Wang [14] adopted CFD simulation results in their stability analysis of flapping wing flight. Gao *et al.* [4] investigated the motion of a model fruit fly with six degrees of freedom based on CFD computation, and simulated passive dynamic motions of the flyer under small perturbation. They argued that an active control with sufficiently fast response is needed to maintain the stability of the flight under disturbance. Kolomenskiy *et al.* [6] simulated taking off flight of fruit-fly with two degrees of freedom using a pseudo-spectral method integrated with a flight dynamics solver.

However, the simulation of insects in free hovering and rectilinear flight and intricate maneuvering sequences remains highly challenging due to their dynamical complexities and high computational cost [17]. General-purpose computing on graphics processing units (GPGPU) is

[1]Dept. of Mechanical Engineering, National University of Singapore, Singapore

an emerging heterogeneous computing technique that performs massive parallelization on graphics processing unit (GPU). This technology is designed to achieve high float point operation rates and is suitable for acceleration of numerically intensive CFD computations. Here, we adopted the NVIDIA CUDA technology to accelerate the simulation of a free flying insect model and investigated its long-term behaviors in different flight scenarios.

1. Numerical Methods

1.1. Governing Equations and Numerical Discretization

The dynamics of the fluid driven by flying insects is governed by the incompressible non-dimensional Navier-Stokes (NS) equations, given here in an arbitrary Lagrangian-Eulerian (ALE) form:

$$\partial_t \mathbf{u} + (\mathbf{u} - \mathbf{u}^g) \cdot \nabla \mathbf{u} = -\nabla p + \frac{1}{\mathrm{Re}} \nabla^2 \mathbf{u}, \tag{1a}$$

$$\nabla \cdot \mathbf{u} = 0, \tag{1b}$$

where $\mathbf{u}$ and p are the non-dimensional velocity and pressure fields respectively of the time-varying fluid domain , and $\mathbf{u}^g$ denotes the convection velocity of the computational node. The convection velocity $\mathbf{u}^g$ for meshfree nodes follow the movement of the boundary surfaces.

An implicit form projection method proposed in [3] is adopted in this work to advance the above governing equations in time. The discretized form of the momentum equations (1a) is written as:

$$\frac{\mathbf{u}^* - \mathbf{u}^n}{\Delta t} = \frac{1}{2} \left\{ \left[-(\mathbf{u} - \mathbf{u}^g) \cdot \nabla \mathbf{u} + \frac{1}{\mathrm{Re}} \nabla^2 \mathbf{u} \right]^{n+1} + \left[-\nabla p - (\mathbf{u} - \mathbf{u}^g) \cdot \nabla \mathbf{u} + \frac{1}{\mathrm{Re}} \nabla^2 \mathbf{u} \right]^{n} \right\}, \tag{2a}$$

$$\frac{\mathbf{u}^{n+1} - \mathbf{u}^*}{\Delta t} = -\frac{1}{2} \nabla p^{n+1}, \tag{2b}$$

where superscripts n and $n+1$ denote time level. The $\mathbf{u}^*$ is an approximation of the velocity field $\mathbf{u}^{n+1}$. Taking the divergence of (2b) yields the following pressure-Poisson equation, which allows us to obtain the new pressure field p^{n+1} from $\mathbf{u}^*$:

$$\nabla^2 p^{n+1} = \frac{2}{\Delta t} \nabla \cdot \mathbf{u}^*. \tag{3}$$

Boundary conditions given in [1], [16] and [17] have been implemented here to close the above discretized governing equations. The fractional step equations and boundary conditions are solved iteratively to advance the flow field to new time level. The pressure-Poisson equation is solved by a hybrid Jacobi-BiCGSTAB solver to attain the solution of the pressure field p^{n+1}.

1.2. SVD-GFD Scheme

Following the methodology developed in [3] and [15], the flow equations are solved on a hybrid Cartesian cum meshfree grid system, wherein the body and wings of the flyer, and their near fluid neighborhood are discretized by meshfree nodes. A set of computational mesh for a fruit fly flyer that used in this study is shown in fig. 1. The meshfree nodes around the boundaries convect with the motion of the body and wings. Generalized finite difference (GFD) scheme is adopted here to approximate derivatives involved in the solution on the meshfree nodes.

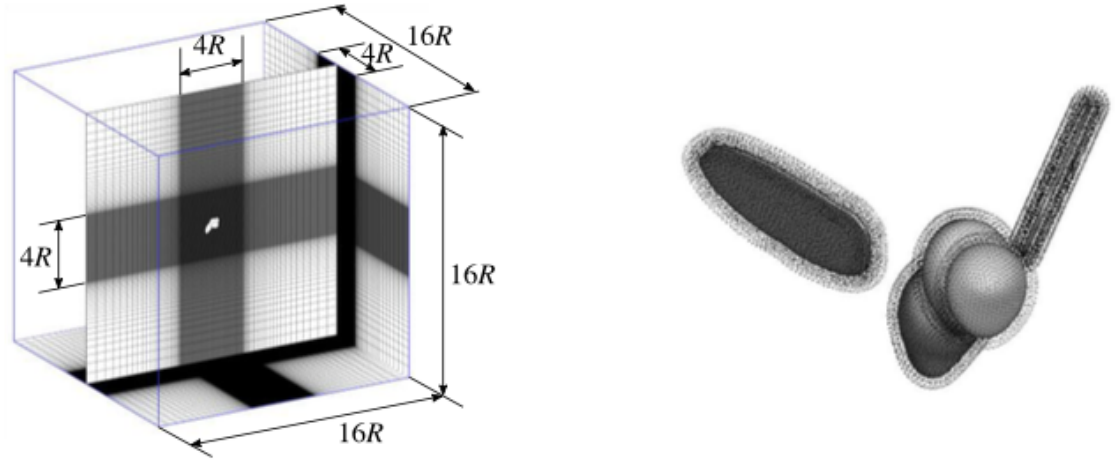


Figure 1. Schematic view of computational domain (R is the wing length of the flyer). Left: background grid; Right: meshfree nodes around insect model

The GFD method is based on the Taylor series expansion. The three dimensional Taylor series expansion of the function $f(\mathbf{x})$ at $\mathbf{x}_1 = \mathbf{x}_0 + \Delta\mathbf{x}_1$ is given in terms of the derivatives at node $\mathbf{x}_0$ to order m by:

$$f(\mathbf{x}_1) = f(\mathbf{x}_0) + \sum_{1\le j_1+j_2+j_3\le m-1} \frac{\Delta x_1^{j_1}\Delta y_1^{j_2}\Delta z_1^{j_3}}{j_1!j_2!j_3!}\left[\partial_x^{j_1}\partial_y^{j_2}\partial_z^{j_3}\right] + \mathcal{O}(|\Delta\mathbf{x}_1|^m), \tag{4}$$

where ∂_x, ∂_y and ∂_z denote the partial derivatives with respect to the coordinate variables.

If the values of the function $f(\mathbf{x})$ are known at n nodes $\mathbf{x}_i = \mathbf{x}_0 + \Delta\mathbf{x}_i$ $(i = 1, 2, \cdots, n)$ in the vicinity of the central node $\mathbf{x}_0$, we can obtain n equations about the derivatives $[\partial_x^{j_1}\partial_y^{j_2}\partial_z^{j_3} f]$ at $\mathbf{x}_0$. By truncating the Taylor series after the third-order derivatives ($m = 4$), the equations can form following linear system:

$$\Delta\mathbf{f}_{n\times 1} = \mathbf{S}_{n\times 19}\partial\mathbf{f}_{19\times 1}, \tag{5}$$

where

$$\mathbf{f}_{n\times 1} = \begin{bmatrix} f_1 - f_0 & f_2 - f_0 & \cdots & f_n - f_0 \end{bmatrix},$$

$$\mathbf{S}_{n\times 19} = \begin{bmatrix} \Delta x_1 & \Delta y_1 & \Delta z_1 & 0.5\Delta x_1^2 & 0.5\Delta y_1^2 & 0.5\Delta z_1^2 & \cdots & \Delta x_1\Delta y_1\Delta z_1 \\ \Delta x_2 & \Delta y_2 & \Delta z_2 & 0.5\Delta x_2^2 & 0.5\Delta y_2^2 & 0.5\Delta z_2^2 & \cdots & \Delta x_2\Delta y_2\Delta z_2 \\ \vdots & & & & & & \ddots & \vdots \\ \Delta x_n & \Delta y_n & \Delta z_n & 0.5\Delta x_n^2 & 0.5\Delta y_n^2 & 0.5\Delta z_n^2 & \cdots & \Delta x_n\Delta y_n\Delta z_n \end{bmatrix},$$

$$\partial\mathbf{f}_{19\times 1} = \begin{bmatrix} \partial_x & \partial_y & \partial_z & \cdots & \partial_x\partial_y\partial_z \end{bmatrix}^T f|_{\mathbf{x}_0}.$$

Should the matrix $\mathbf{S}$ be a non-singular square matrix, an approximate solution of the derivatives $\partial\mathbf{f}_{19\times 1}$ at the central node may be determined by $\partial\mathbf{f}_{19\times 1} = [\mathbf{S}_{n\times 19}^{-1}]\Delta\mathbf{f}_{n\times 1}$ with formal accuracy of $\mathcal{O}(|\Delta\mathbf{x}|^3)$ and $\mathcal{O}(|\Delta\mathbf{x}|^2)$ for first order and second order derivatives, respectively. This involves finding the pseudo-inverse of the coefficient matrix $\mathbf{S}$. However, $\mathbf{S}$ tends to be ill-conditioned due to irregular arrangements of the neighboring nodes, especially when two or more nodes in the neighborhood are located very close to each other, making its inversion impossible to be implemented in practice.

The least-squares method and singular value decomposition (SVD) technique are classical approaches that give the best-fitting solution/approximation for over-determined systems. In the present 3D simulation, we implemented the SVD-GFD method presented by [1]. In the algorithm, a regularization of the solution by setting the very small singular values of $\mathbf{S}$ to zero is performed in order to avoid ill-conditioning. The system may become under-determined when the small singular values were omitted, but the SVD method ensures a solution with minimum L_2 error can be obtained. The SVD scheme with regularization has been found to be robust and accurate in the practice of 3D simulations for natural flyers and swimmers [11, 17, 18].

1.3. Nodal Selection

To minimize the approximation error caused by arbitrary node distribution in the meshfree cloud, we adopted a minimum of 40 supporting nodes in the computation of $\partial \mathbf{f}_{19\times 1}$ for the central node and applied a novel nodal selection criterion to improve the configuration of the supporting nodes. In the current scheme, all neighboring nodes inside a predetermined support region are collected as candidates for supporting nodes and are sorted in distance order, from the closest to the furthest. A sphere with radius r_{inf} and a cone with angle α are assigned to each node including the central one as its zone of influence (fig. 2):

$$r_{inf} = \begin{cases} r_{inf0}, & \text{for central node,} \\ r_0 \sin\alpha, & \text{for nodes with } \bar{r}_i \le r_0, \\ \bar{r}_i \sin\alpha, & \text{for nodes with } \bar{r}_i > r_0, \end{cases} \tag{6}$$

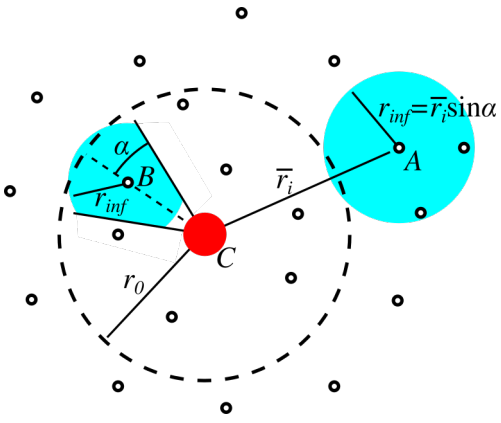


Figure 2. Schematic drawing showing region of influence of nodes A and B in the neighborhood of central node C

Moreover, it was pointed out that all the supporting nodes should be visible to the central node to ensure that the derivatives can be correctly discretized [11]. As defined in geometry, a supporting node is visible to the central node if the line segment that connects them does not intersect any boundary surface. A visibility check base on ray-casting algorithm is applied on the nodes close to a sharp edged object or a thin air foil like the insect wing, as their candidate nodes are probably selected across the boundary surface. As shown in fig. 3, the selected supporting nodes satisfy the visibility criterion for nodes near solid surface.

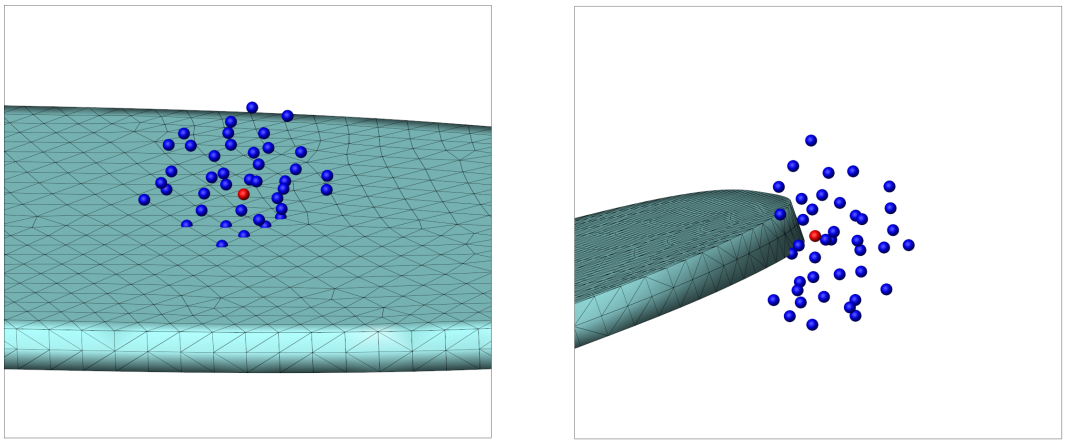


Figure 3. Resultant supporting nodes (blue) of central nodes (red) near solid surface

2. Implementation

The motion of the model insect in free flight is motivated by aerodynamic forces and is subject to Newton's law. With the rigid body assumption, a flapping wing flyer in free flight can be treated as a multi-body system that has six degrees of freedom (6-DoF). The wing mass of fruit fly (*D. melanogaster*) is about several micro grams and contributes to less than 0.5% of the total mass [2]. Hence, we here ignored the wing mass of the model fruit fly in the present study. Moreover, we supposed that the density of insect body is homogeneous, and computed the fly's inertia tensor and centre-of-mass (CoM) based on this homogeneous density distribution. The equations of motion for the 6-DoF flight are given by:

$$\begin{cases} m\dot{\mathbf{V}}_C(t) = m\mathbf{g} + \mathbf{F}_A(t) \\ \dot{\mathbf{X}}_C(t) = \mathbf{V}_C(t) \\ \dot{\mathbf{L}}_C(t) = \mathbf{M}_A(t) \\ \dot{\mathbf{\Theta}}_C(t) = \mathbf{K}(t) \cdot \boldsymbol{\omega}_C(t) \end{cases}, \tag{7}$$

where $\mathbf{V}_C$, $\mathbf{X}_C$, $\mathbf{L}_C$ and $\mathbf{\Theta}_C$ denote the linear velocity, body position, angular momentum and body orientation vector of the flyer at its body centre-of-mass (CoM) C in the global frame, respectively, $\mathbf{K}$ is a transformation matrix, $\boldsymbol{\omega}_C$ is the flyer's angular velocity determined by its angular momentum and moment of inertia, $\mathbf{g}$ is the gravity vector, and $\mathbf{F}_A$ and $\mathbf{M}_A$ are the aerodynamic forces and moments acting on the model fly respectively.

The CFD scheme presented in the last section was adopted here to resolve the flow field surround the flapping wing flyer, in order to model the aerodynamic forces from the pressure and viscous stress acting on the flyer surfaces. The surface configuration $\Gamma(\mathbf{G}(t)|\mathbf{V}_C, \mathbf{X}_C, \boldsymbol{\omega}_C, \mathbf{\Theta}_C)$ is time-dependent and comprise information about the geometry of the model fly (body and wings) $\mathbf{G}(t)$ in its body frame as well as the information on the state of motion $\boldsymbol{\xi} = [\mathbf{V}_C, \mathbf{X}_C, \boldsymbol{\omega}_C, \mathbf{\Theta}_C]^T$ of the flyer in the global frame. $\Gamma(t)$ determines the boundary condition in the CFD computation while is in turn solved from (7) using the resultant aerodynamic forces. Hence, the solution of $\Gamma(t)$ involves fluid-body interaction (FSI) and is essentially an implicit problem.

This FSI problem was solved through a fixed-point iteration on $\Gamma(t)$ using a predictor-corrector method based on the 4-step implicit Adams-Moulton scheme in the present study. The reason why multistep methods were selected is that evaluating aerodynamic forces by the CFD method in intermediate steps requires extra computing time and storage, and the time step

size used in the CFD part is small enough to ensure stability of the selected multistep method. Moreover, the consistent time integration intervals allow the iterative projection method in CFD part to be embedded into the 6-DoF motion solver to achieve a tightly coupled fluid-body interaction simulation.

The calculation procedures are listed below and showed in fig. 4 in details:

1. Predicator step (P): at time step $n+1$, compute the current dynamic state of insect $\boldsymbol{\xi}^{n+1}$ from the previous dynamic state using the 2-step Adams-Bashforth scheme:

$$\boldsymbol{\xi}^{n+1,0} = \boldsymbol{\xi}^{n} + \Delta t \left(\frac{3}{2}\dot{\boldsymbol{\xi}}^{n} - \frac{1}{2}\dot{\boldsymbol{\xi}}^{n-1} \right); \tag{8}$$

2. Evaluator step (E): update the flow field state Φ from Φ^n to Φ^{n+1} using the CFD solver with boundary conditions given by $\boldsymbol{\xi}^{n+1}$, then evaluate $\dot{\boldsymbol{\xi}}^{n+1}$ from the latest available solution of Φ;
3. Corrector step (C): correct $\boldsymbol{\xi}^{n+1}$ using the 4-step implicit Adams-Moulton scheme:

$$\boldsymbol{\xi}^{n+1,j} = \boldsymbol{\xi}^{n} + \Delta t \left(\frac{3}{8}\dot{\boldsymbol{\xi}}^{n+1,j} + \frac{19}{24}\dot{\boldsymbol{\xi}}^{n} - \frac{5}{24}\dot{\boldsymbol{\xi}}^{n-1} + \frac{1}{24}\dot{\boldsymbol{\xi}}^{n-2} \right). \tag{9}$$

The above algorithm ran in iterative P(EC)^{k}E mode, with one predictor step and k corrector iterations, to solve the rigid body dynamics of insect flight.

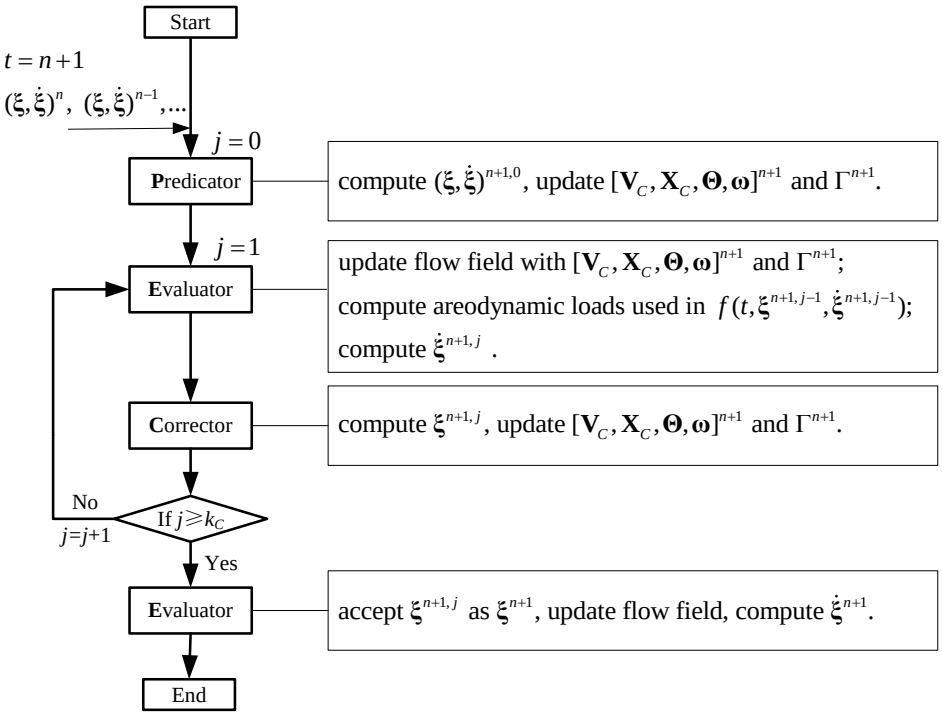


Figure 4. Schematic of predicator corrector solution procedure in one time step

The CFD solution in the Evaluator step was orders of magnitude more numerically intensive than other calculations in the present in-house code. Hence, the in-house OpenMP parallelized code was re-programmed with CUDA technology to exploit the capability of modern heterogeneous clusters.

Parallelization of the CFD computations was straight forward due to semi-implicit nature of the projection method given by Equation (2a) and (2b), and the BiCGSTAB method for

the pressure-Poisson equation (3). However, the SVD algorithm executed on each meshfree node required heavy numerical workload on individual one and was not yet re-programmed into reliable and efficient GPU code. However, the SVD algorithm executed on each meshfree node generated significant numerical workload that has still not yet been re-programmed into reliable and efficient GPU code. Hence, the SVD calculation of the CFD solver was performed on CPUs only in present simulations and was one aspect of the computation that could be further improved upon.

3. Results and Discussion

3.1. Benchmarking Cases

The parallel code was performed on the heterogeneous cluster of the National Supercomputing Centre of Singapore with the NVIDIA Tesla K40 accelerator installed. Additionally, an Intel Xeon E5 Workstation with the NVIDIA Tesla K20c accelerator was used for debugging and benchmarking purposes. Benchmarking computations were conducted on the Xeon E5 Workstation using three different mesh systems shown in tab. 1 (Mesh size: Set 1<Set 2<Set 3). From fig. 5a-c, it can be seen that the parallelization and the GPU acceleration greatly speedup the nodal search process and fractional step iteration in above CFD scheme. The GPU speedup increased with the size of the Cartesian mesh, because time used for data transfer between CPU and GPU became less important with increasing computation time. The drop of speedup in node search may be explained by the increasing overhead of memory operand in larger array. The time for the projection method calculation with GPU acceleration is about half of computation using only 12 threads of CPU parallelization. Considering the wall-time elapsed during one complete FSI iteration, the advantage of GPU acceleration is more impressive when the grid size is large (fig. 5d), due to the computational bottleneck caused by the SVD procedure executing on CPUs. In general, the current GPU-accelerated solver allowed us to complete the simulation of about 100 wingbeats of the free flying insect in forward flight within a week time. This significant reduction of computational time made possible by the using of GPUs allowed us to obtain the long-term quasi-steady state of the model insect in free forward flight.

Table 1. Details of benchmarking mesh sets

	Set 1	Set 2	Set 3
Cartesian grid	$125 \times 125 \times 245$	$161 \times 161 \times 161$	$221 \times 221 \times 221$
Meshfree nodes	40039	31863	31863

We adopted the experimental results in [10] to validate the CFD scheme presented in this paper. In the experiments of Muijres *et al.* [10], the forces on the insect wings were estimated from a scaled robot wing and normalized to the fly scale using $F^* = F/mg$ with a body mass of $m = 1.8$ mg. Compared to their unfiltered experimental data, our numerical results closely tracked the build-up and decrease of forces, and correctly captured major force peaks and troughs in the whole wingbeat (see fig. 6). The mean lift obtained in the present CFD simulation was 13.5% higher than the experimental results, while a 9.7% surplus was obtained on the mean drag. This relative error between experimental and numerical results agreed with the previous numerical studies [9, 11, 17], and it may caused by the oscillation/flutter of the robotic wing due to its imperfect rigidity and slips within the actuator mechanisms. This agreement indicates

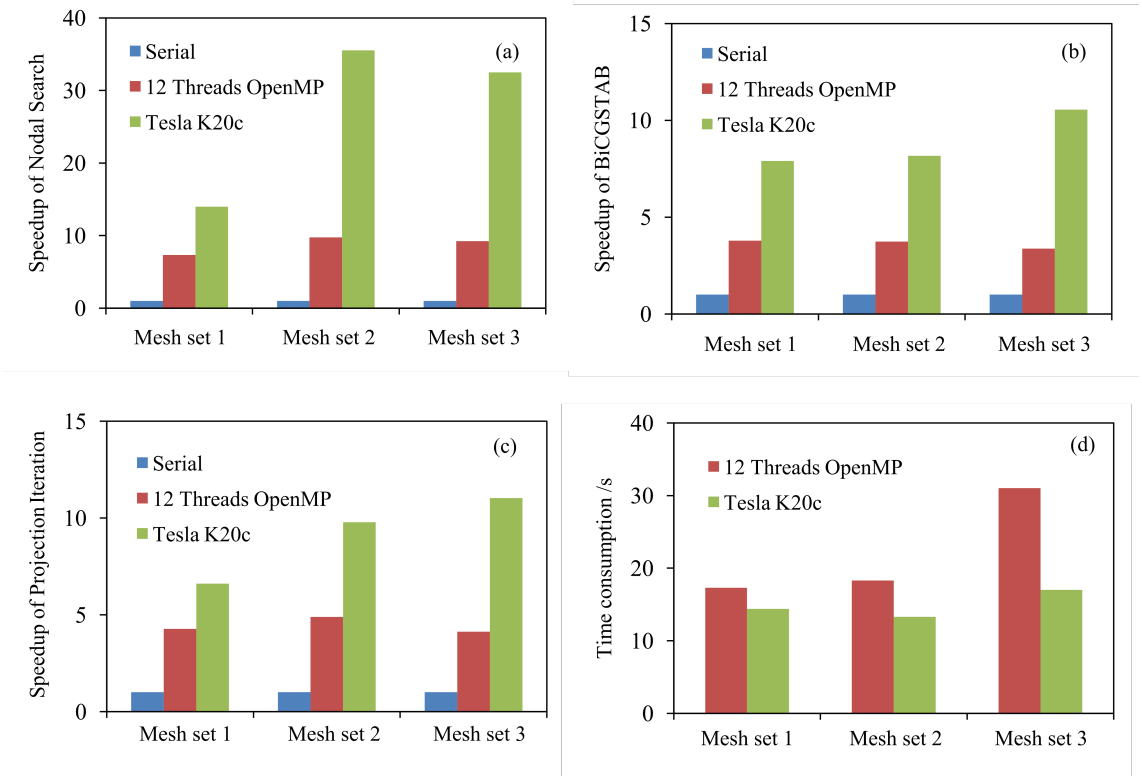
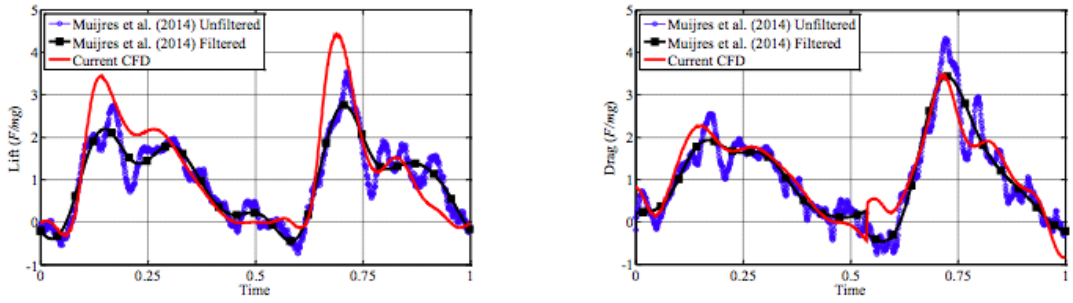


Figure 5. Performance of parallel code

that the present GPU-accelerated CFD solver can predict the force generation of insect wings with sufficient accuracy for our purpose.



(a) Lift force. (b) Drag force.

Figure 6. Comparison of computational forces of a fruit fly wing executing natural wing motion at Re=115 with experimental results from [10]

3.2. Flight Simulations

The present numerical method resolves the full details of temporal dynamics of the flow field, and provides us a convenient way to visualize and quantify the 3D flow field produced by the flapping wings.

In hovering flight, the wings shed a copious amount of vorticity into the surrounding air. These took the forms of a leading-edge vortex (LEV), a wing-tip vortex (WTV) and a trailing-edge vortex (TEV) - identified by the regions on the wing where they are generated as shown fig. 7a. The vortices connected with each other to form a vortex ring (VR) on the wings

. The present results show that there existed a strong spanwise flow on the flapping wing in hovering flight. As suggested by previous studies [5, 7], the spanwise flow that may drain energy from the LEV, and thus limited its downstream chordwise development and prevented the wings from stalling. However, the increase of non-dimensional spanwise velocity, u^*_{span}, from the wing root to the middle wing-span also suggests that the spanwise flow may stretch the LEV along the wing and hence helped to prolong the attachment of LEV [8]. In fact, as u^*_{span} decreased in the distal section of the wing, the LEV soon detached from the wing surface. The spanwise flow may also confine the development of the TEV near the wing root which formed sheet-like vortices.

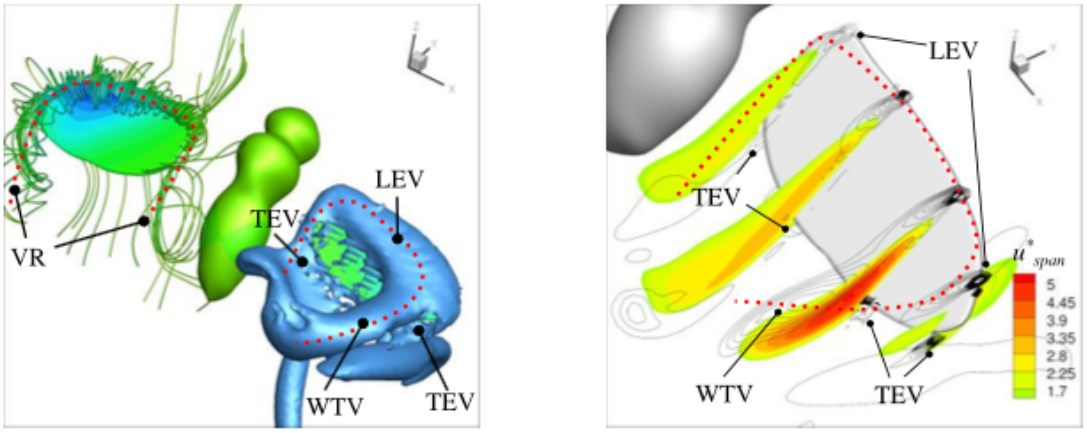
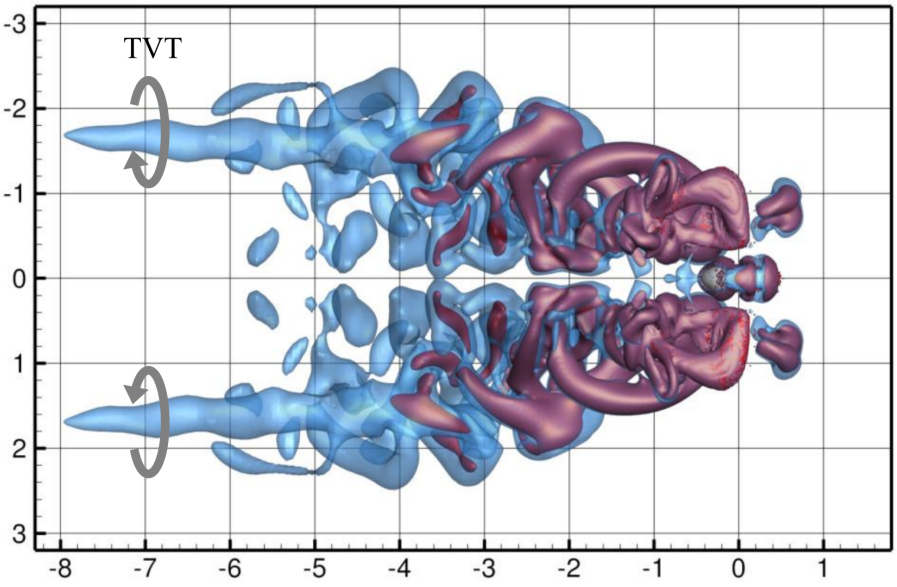


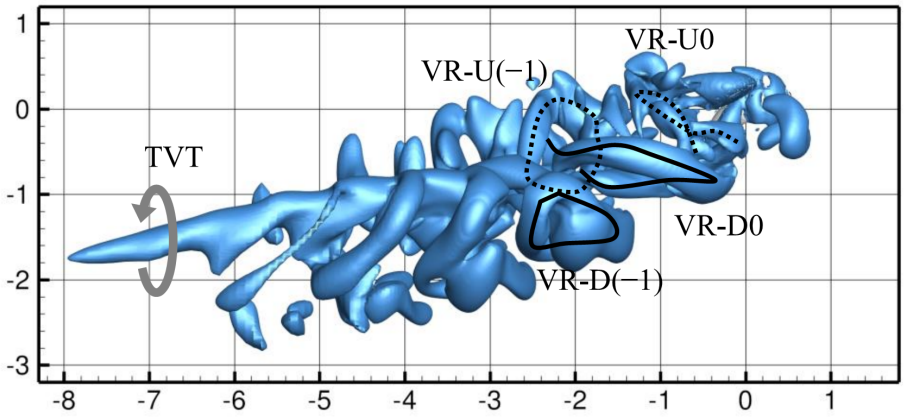
Figure 7. Vortices on the flapping wing obtained in hovering flight. Left: stream traces and iso-surface of $\lambda^*_2 = -80$; Right: u^*_{span} contours and λ^*_2 iso-lines

In forward flight, the wing stroke plane of the flyer is typically rotated forward to generate the requisite wing thrust, while the body is pitched forward to reduce aerodynamic drag. The quasi-steady posture of the flyer and the stroke angle are speed-dependant and ultimately governed by the balance of forces and moments acting on the whole flyer. Overall, the bulk of thrust was produced during the upstroke of a wingbeat, while the bulk of lift was generated by the downstroke. As shown in fig. 8, the vortex wake behind model fruit flyer in 60 cm/s flight - the near wake contains vortex rings shed by the up- and down-strokes, which eventually decayed to form a pair of travelling vortex tube (TVT) further downstream.

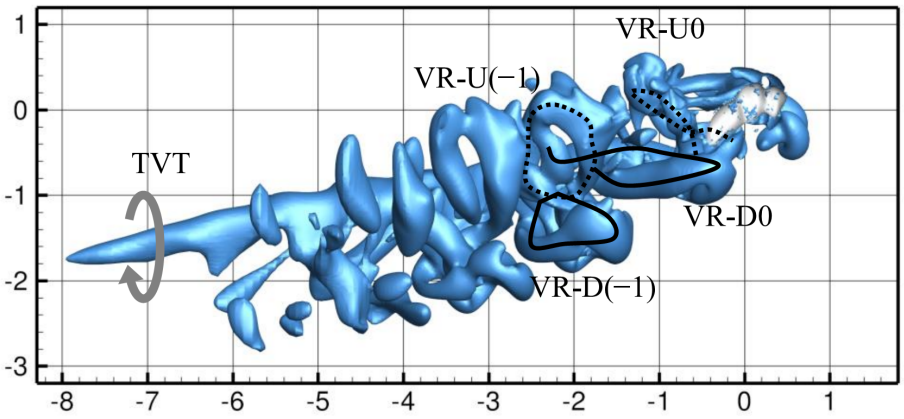
The shedding vortices become even more complex when the insect flyer is executing fast maneuvers, as the large body motion of the flyer interfered directly with the shed vortices in the wake. A simulation of fast banking flight has been carried out in this work. The insect was controlled to turn to its right side from steady hovering state in the banking flight. As shown in fig. 9, the wings of the rolling model fly collided with the stacked VRs and broke them down into disconnected vortex tubes. There were two highly stretched vortex tubes existing in the wake. During the fast rightward rolling, a strong vortex ring was created in the wake below the left wing (see fig. 9c) and the vortex moved towards the left side of the flyer (fig. 9b d). This vortex ring conveyed the lateral momentum induced by the rolling motion, and is noted as the rolling vortex. Thereafter, the downstroke WTV created on the right wing was greatly elongated to form a yawing vortex during the fast yawing phase, and remained visible in the flow field till the fifteenth wingbeat (fig. 8e f). The transient structure of the vortex wake soon decayed and evolved into the normal hovering wake after the flyer re-stabilized itself in a new hovering state.



(a) Top view.



(b) Side view



(c) Section view taken from the sagittal plane.

Figure 8. Shedding vortices (VR-Dn (—) and VR-Un ($\cdots$)) on 80 cm/s forward flight showing by λ_2^* iso-surfaces (light blue $\lambda_2^* = -0.18$, dark red $\lambda_2^* = -1.8$)

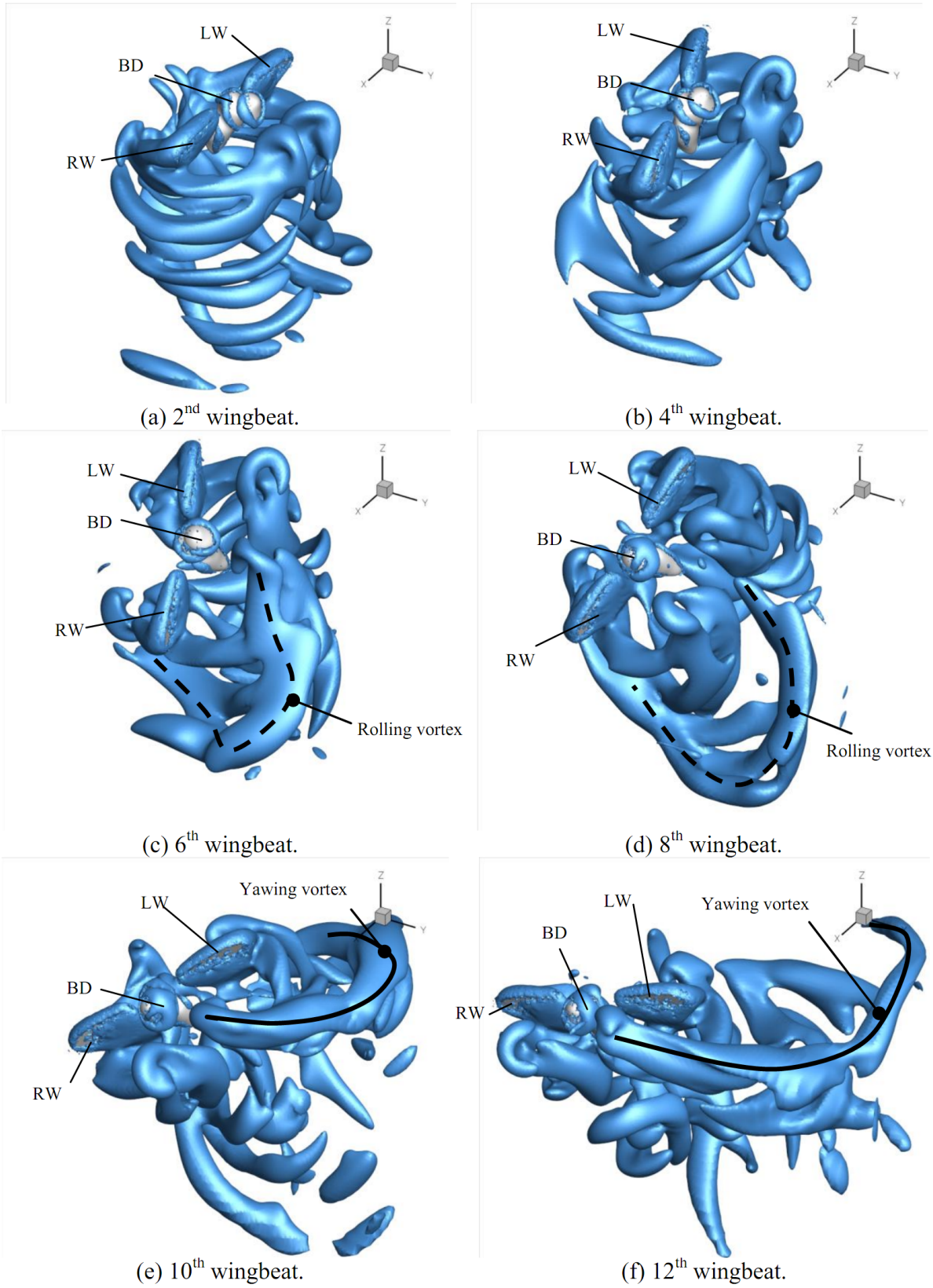


Figure 9. Development of vortex wake in banking flight (banking from steady hovering), showing by $\lambda_2^* = -1.8$ iso-surfaces obtained at the mid-downstroke. (BD: insect body; LW: left wing; RW: right wing)

Conclusions

This paper presents a numerical study on the flapping-wing free flight of a model fruit-lfy (*D. melanogaster*). The simulations were carried out with a coupled CFD/6-DoF motion solver on a 3D Cartesian-cum-meshfree grid by an SVD-GFD computational scheme. The study covered a variety of flight scenarios including hovering, forward flight and banking maneuvers. Unlike most previous insect flight studies, the model insect was allowed to move freely in all the six degrees of freedom. The computationally-intensive simulations were expedited by the application of CUDA-based GPUs, which greatly accelerated the speed of the CFD (Navier-Stokes) solver over that of the original CPU parallelized code. The gains in turnaround time were particularly significant and beneficial of highly prolonged simulations carried out to investigate the long-time quasi-steady performance of certain flight states, such as hovering and long-distance forward flights. The simulations allowed us to probe the complex aero-cum-body dynamics in flapping wing insect flight, and to study and refine control strategies and algorithms to achieve steady flight and complex aerial maneuvers.

The present computational simulations reveal the complex vortical wakes created by the wings of the model insect. The strong LEV, TEV, and WTV shed by the flapping wings dominated the highly complex near-wake of the flyer. An even more complex wake system enveloped the model insect in sharp maneuvering flights, where the wings may directly interfere with the shed flow structures. These flow structures are governed by the kinematics of the flapping wings and the motion of the flyer; and their analyses will help us to better understand the underlying physics. In level forward flight, the vortex wake in the rear of the model insect may decay into a pair of trailing vortices, similar to those frequently observed behind airplanes in flight.

The authors wish to thank the High Performance Computing Centre of National University of Singapore and National Supercomputing Centre of Singapore for providing their high performance computing facilities.

References

1. Ang, S.J., Yeo, K.S., Chew, C.S., Shu, C.: A singular-value decomposition (SVD)-based generalized finite difference (GFD) method for close-interaction moving boundary flow problems. International Journal for Numerical Methods in Engineering 76(12), 1892–1929 (2008), DOI:10.1002/nme.2398

2. Bergou, A.J., Ristroph, L., Guckenheimer, J., Cohen, I., Wang, Z.J.: Fruit Flies Modulate Passive Wing Pitching to Generate In-Flight Turns. Physical Review Letters 104(14) (Apr 2010), DOI:10.1103/physrevlett.104.148101

3. Chew, C., Yeo, K.S., Shu, C.: A generalized finite-difference (GFD) ALE scheme for incompressible flows around moving solid bodies on hybrid meshfree-cartesian grids. Journal of Computational Physics 218(2), 510–548 (2006), DOI:10.1016/j.jcp.2006.02.025

4. Gao, T., Lu, X.Y.: Insect normal hovering flight in ground effect. Physics of Fluids 20(8), 087101 (2008), DOI:10.1063/1.2958318

5. Jardin, T., Farcy, A., David, L.: Three-dimensional effects in hovering flapping flight. Journal of Fluid Mechanics 702, 102–125 (Jul 2012), DOI:10.1017/jfm.2012.163

6. Kolomenskiy, D., Maeda, M., Engels, T., Liu, H., Schneider, K., Nave, J.C.: Aerodynamic Ground Effect in Fruitfly Sized Insect Takeoff. PLOS ONE 11(3), e0152072 (Mar 2016), DOI:10.1371/journal.pone.0152072

7. Lentink, D., Dickinson, M.H.: Biofluiddynamic scaling of flapping, spinning and translating fins and wings. Journal of Experimental Biology 212(16), 2691–2704 (Aug 2009), DOI:10.1242/jeb.022251

8. Lim, T.T., Teo, C.J., Lua, K.B., Yeo, K.S.: On the prolong attachment of leading edge vortex on a flapping wing. Modern Physics Letters B 23(03), 357–360 (2009), DOI:10.1142/s0217984909018394

9. Liu, H.: Integrated modeling of insect flight: From morphology, kinematics to aerodynamics. Journal of Computational Physics 228(2), 439–459 (2009), DOI:10.1016/j.jcp.2008.09.020

10. Muijres, F.T., Elzinga, M.J., Melis, J.M., Dickinson, M.H.: Flies evade looming targets by executing rapid visually directed banked turns. Science 344(6180), 172–177 (2014), DOI:10.1126/science.1248955

11. Nguyen, T.T., Shyam Sundar, D., Yeo, K.S., Lim, T.T.: Modeling and analysis of insect-like flexible wings at low Reynolds number. Journal of Fluids and Structures 62, 294–317 (Apr 2016), DOI:10.1016/j.jfluidstructs.2016.01.012

12. Shyy, W., Aono, H., Kang, C.K., Liu, H.: An introduction to flapping wing aerodynamics. Cambridge aerospace series, Cambridge University Press, Cambridge; New York (2013), DOI:10.1017/cbo9781139583916.007

13. Shyy, W., Kang, C.K., Chirarattananon, P., Ravi, S., Liu, H.: Aerodynamics, sensing and control of insect-scale flapping-wing flight. Proceedings of the Royal Society A: Mathematical, Physical and Engineering Science 472(2186), 20150712 (2016), DOI:10.1098/rspa.2015.0712

14. Sun, M., Wang, J.K.: Flight stabilization control of a hovering model insect. Journal of Experimental Biology 210(15), 2714–2722 (Aug 2007), DOI:10.1242/jeb.004507

15. Wang, X.Y., Yeo, K.S., Chew, C.S., Khoo, B.C.: A SVD-GFD scheme for computing 3d incompressible viscous fluid flows. Computers & Fluids 37(6), 733–746 (2008), DOI:10.1016/j.compfluid.2007.07.022

16. Wang, X.Y., Yu, P., Yeo, K.S., Khoo, B.C.: SVDGFD scheme to simulate complex moving body problems in 3d space. Journal of Computational Physics 229(6), 2314–2338 (2010), DOI:10.1016/j.jcp.2009.11.037

17. Wu, D., Yeo, K.S., Lim, T.T.: A numerical study on the free hovering flight of a model insect at low reynolds number. Computers & Fluids 103, 234–261 (2014), DOI:10.1016/j.compfluid.2014.07.030

18. Yu, P., Yeo, K.S., Shyam Sundar, D., Ang, S.J.: A three-dimensional hybrid meshfree-Cartesian scheme for fluid-body interaction. International Journal for Numerical Methods in Engineering 88(4), 385–408 (Oct 2011), DOI:10.1002/nme.3182

DOI: 10.14529/jsfi170203

Simultac Fonton: A Fine-Grain Architecture for Extreme Performance beyond Moore's Law

Maciej Brodowicz[1], *Thomas Sterling*[1]

With nano-scale technology and Moore's Law end, architecture advance serves as the principal means of achieving enhanced efficiency and scalability into the exascale era. Ironically, the field that has demonstrated the greatest leaps of technology in the history of humankind, has retained its roots in its earliest strategy, the von Neumann architecture model which has imposed tradeoffs no longer valid for today's semiconductor technologies, although they were suitable through the 1980s. Essentially all commercial computers, including HPC, have been and are von Neumann derivatives. The bottlenecks imposed by this heritage are the emphasis on ALU/FPU utilization, single instruction issue and sequential consistency, and the separation of memory and processing logic ("von Neumann bottleneck"). Here the authors explore the possibility and implications of one class of non von Neumann architecture based on cellular structures, asynchronous multi-tasking, distributed shared memory, and message-driven computation. "Continuum Computer Architecture" is introduced as a genus of ultra-fine-grained architectures where complexity of operation is an emergent behavior of simplicity of design combined with highly replicated elements. An exemplar species of CCA, "Simultac" is considered comprising billions of simple elements, "fontons", of merged properties of data storage and movement combined with logical transformations. Employing the ParalleX execution model and a variation of the HPX+ runtime system software, the Simultac may provide the path to cost effective data analytics and machine learning as well as dynamic adaptive simulations in the trans-exaOPS performance regime.

Keywords: High-Performance Computing, parallel processing, exascale, non-von Neumann architecture, cellular architecture.

Introduction

Commercial computers have been predominantly von Neumann derivative architectures throughout the seven decades of digital electronic information processing although the enabling technologies and the logical structures have varied widely over this period. Such systems like MPPs, SIMD, vector, ILP, and multithreaded, while representing their own distinct ways of exploiting parallelism, have none the less been based on the fundamental principles of the von Neumann model of the 1940s. These include sequential instruction issue and sequential consistency, optimizing for ALU/FPU utilization as the precious resource, and separation of processing and memory. At one time this was the rational objective function as an FPU in the enabling technology of their respective times was the most expensive component in discrete components, size, cost, and power; thus, driving its throughput as most important for performance to cost. Even when this was less so, while Moore's Law dominated, component capacity and performance grew exponentially with time in conjunction with Dennard scaling [7] and ILP through incremental extensions of conventional practices and architecture. Since 2005, processor core speeds have not grown and overall system performance has been improved only through increased multiplicity of cores either through multicore/manycore architectures such as the Intel Xeon Phi or the higher order core structures like the NVIDIA family of GPUs. This will likely deliver an ExaFLOPS HPL R_{max} performance after 2020 but at significant cost and low efficiency. Further emerging applications of importance in data analytics and machine learning among others will demand very different design structures, balance, and programming methodologies. It is a

[1]Center for Research in Extreme Scale Technologies, Indiana University, Bloomington, USA

premise of this paper that the future of high end computing exploiting CMOS nano-scale device technologies will require new classes of computer architecture to take full advantage of component capabilities through avoidance of many of the bottlenecks imposed by the von Neumann architecture model. One example of active interest and research pursuit is neuro-morphic architectures that are brain-inspired such as neural nets. The research discussed here, Continuum Computer Architecture (CCA), is proposed to reverse the stagnation of von Neumann architecture and dramatically increase key properties of scalability and memory bandwidth while dramatically reducing size, power, and cost. This paper establishes the fundamental principles of the future class CCA systems.

CCA is motivated by the end of Moore's Law at nanoscale semiconductor feature size and the opportunity for new architectures that exploit rational optimization strategies of current enabling technologies as opposed to conventional practices based on the legacy of the venerable von Neumann architecture model. An examination of conventional core architectures based on the three major contradictions imposed by the von Neumann model are the emphasis on FPU utilization, the separation of processor and memory, and the limited way in which parallelism is exposed and exploited due to sequential instruction issue. Much of the die area is dedicated to emphasizing the FPU/ALU utilization including speculative execution, branch prediction, out of order completion, and multiple cache layers and their control. The principal metric of operation should be the time delay between when an operation is logically ready to be executed (that is satisfies its precedent constraints) and when it is actually performed. Eliminating the von Neumann bottleneck associated with the latency and bandwidth constraints for memory access has been long recognized as an essential goal but current cache based techniques demand data reuse through temporal and spatial locality. The typical practice of organizing parallel computation through the BSP style [21] with global barriers imposes potentially severe bottlenecks in core usage with irregular tasks. Furthermore, both user productivity and performance portability are hampered by the myriad details of performance optimization resulting from the complexity of contemporary core designs. Ultimately, future architectures across the exascale performance regime will depend how they address the key fundamental operational factors of starvation, latency, overhead, and contention at every level with effective use of parallelism probably most critical.

This paper departs from the norm by considering these key issues and examining an architecture design space that falls into the broad family of non-von Neumann architectures to eliminate the deleterious effects of the legacy of von Neumann based machines. It further presents a genus of innovative architectures, Continuum Computer Architectures, that borrows from early consideration of a number of alternative strategies including cellular automata [14], dataflow [8], systolic arrays [5], and more recent work on Asynchronous Multi-Task computing [10, 12, 15, 20, 23] as well as the authors' own work on the ParalleX execution model [1, 11]. The paper concludes with an analysis that suggests that an exascale computer employing such concepts could be devised and fabricated using contemporary CMOS semiconductor technology and managed through current experimental dynamic adaptive software techniques in a relatively cost effective way compared to current projected approaches.

1. Continuum Computer Architecture

The architecture genus of Continuum Computer Architecture is first advanced as a Gedanken-experiment with the following attributes: presume that a finite space (rectangular

for convenience) is filled with many rows and columns of computers connected in a mesh topology. Within the finite space, replace each computer with four machines, each of a quarter lower performance, data storage, and foot print so that the same area has the same capability, just realized through more and more smaller and smaller computing elements. Repeat this cycle infinitely. In the limit, every point in a finite space has the properties of logical operations, data storage, and data movement but to a minimal, actually zero amount, in each case. But a finite area, no matter how miniscule beyond a point will have non-zero values in all three parameters. A medium of computing has been created, at least of the imagination a continuum computer. Such continuum computers can actually be produced for special purposes in the analog domain such as the determination of electrical fields in a conductive fluid; not exactly programmable in a general purpose sense but it does prove the point. For a more general CCA, such a medium must be discretized as is done with many algorithmic models such as Discrete Fourier Transforms [6] or Finite Element Methods [2]. The challenge is to derive the smallest fine grain element that embodies the properties of data storage, logic, and data transfer such that when employed in collectives can accomplish real world (programmable) parallel computation. In the special case of the species of CCA described in this paper, the element is referred to as a "fonton", but this is getting ahead of the story, to which we return in future sections. It is ironic that perhaps the first example of this genus of non-von Neumann architectures was created by John von Neumann himself in 1949 in the form of the cellular automata which he proved to be Turing equivalent. Many special purpose cellular automata have been devised over ensuing years; the most famous of which is most likely Conway's Game of Life [3]. The technical strategy of the CCA is summarized as comprising the following elements:

1. Non von Neumann Architecture – to eliminate the bottlenecks imposed by legacy structures.
2. Optimizing for most expensive property, not FPU utilization – emphasizing today's crucial bottlenecks such as memory bandwidth and latency rather than the obsolete notion of FPU utilization.
3. Cellular Structures -- maximizing best usage of die area through cellular structures which replaces complexity of design for complexity of operation with simplicity of design combined with massive replication.
4. Nearest Neighbor Access – exploitation of nearest neighbor communication between cells for extreme system bandwidth with minimum latency as appropriate.
5. Parallel Control Flow – the replacement of sequential issue based control flow with intrinsic parallel control flow combining dataflow and futures semantics with a high level global parallel control state.
6. Objective Function – parameter set for a multi-dimensional optimization space including memory bandwidth and latency, delay between enabled and executed actions, sustained OPS versus peak or Linpack FLOPS, and time to solution versus cost.

These strategy elements in concert establish the guidelines that govern the manifestation of continuum computer architectures for ultra-scale computing.

2. Simultac Fonton

The Simultac architecture currently investigated by the authors is but one of possible species of CCA. It stresses practical aspects of the design to promote cost-effective and efficient implementation while reducing the design time and enabling fast prototyping on FPGA technology.

The primary guiding principle is simplicity; the complexity of its component cells is significantly lower than that of a single RISC core. The desired performance level of the whole system is achieved through high level of replication of uniform component cells (fontons). The emergent properties of a large scale system manifest themselves due to synergies arising within the multitude of concurrent actions performed by many fontons.

Simultac design directly addresses performance degradation factors identified by the SLOWER model [16]. It supports hierarchy of parallel actions, ranging from dataflow-like interactions inside cell groups to parallel process instances, each occupying a macroscopic fraction of hardware resources. This limits starvation by identifying opportunities for parallelism extraction at multiple levels of program execution. The latency effects are suppressed by introduction of new structures, both in software and hardware that enable opportunistic scheduling of work. The overheads are minimized through development of optimized hardware mechanisms; due to simplicity of the design, the analysis of costs and implementation of such improvements are not difficult to perform. The contention is addressed by providing abundant execution resources. A critical element of reducing power consumption, proximity of data, is achieved through integration of storage and processing logic. Finally, component redundancy and high availability "anywhere in the medium" provides fault tolerance.

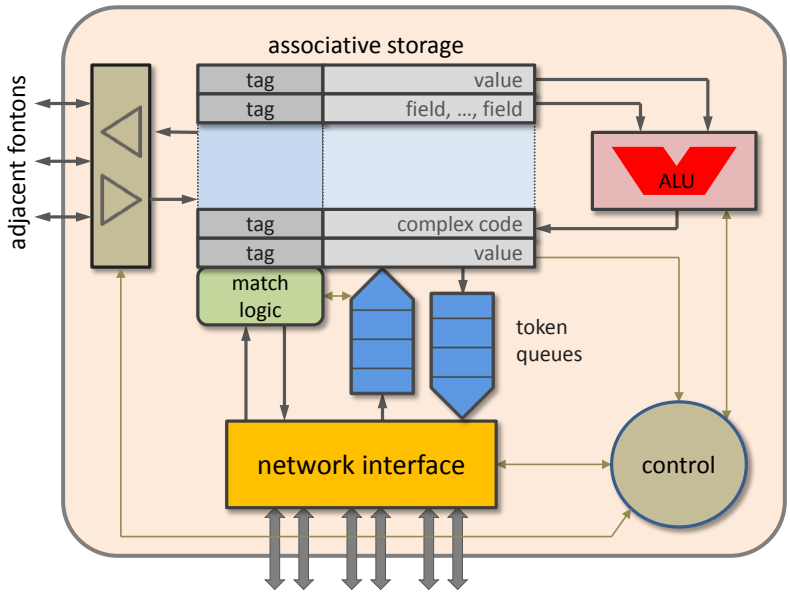


Figure 1. Internal structure of a fonton

A schematic diagram of fonton internals is shown in fig. 1. At the heart of the fonton is an associative storage array in which memory segments with capacity comparable to that of a cache line are combined with virtual tags to enable selection of the appropriate individual storage cells. Contents of the array may be transformed by an ALU controlled by logic executing locally stored strands of instructions. The (limited) code may be either kept in the memory array or delivered by an incident token (active message) from the cross-chip interconnect. The token destination address is matched by specialized logic monitoring all locally used tags. If a match is obtained, including forms of wildcard addressing, the token is absorbed by a fonton. The fonton has a dedicated adjacency interface permitting access and modification of the state of several neighbor fontons. This interface also permits realization of simple but high aggregate bandwidth massaging, alleviating the load of on-chip network if only intra-neighborhood communication is

necessary. The majority of fonton operations are performed within a single clock cycle eliminating elaborate pipelines and reducing the processing latency.

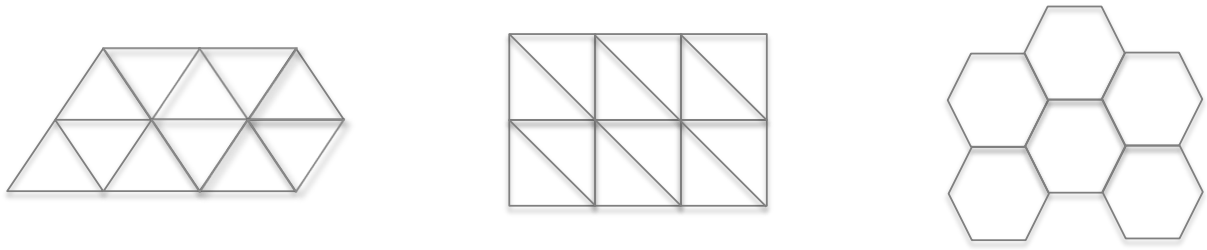


Figure 2. Some of the possible 2D tessellations

Scaling to chip size involves embedding a large number of fontons on a die. For this purpose, die area has to be tessellated using uniform shapes to completely fill the available space. This also determines the number of adjacent fontons involved in the nearest-neighbor interactions. Some of the considered tessellations are illustrated in fig. 2. The resultant geometry directly impacts the types of parallel processing that can be performed in local neighborhoods. They range from simple 1D pipelines, forked pipelines, overlapping multi-directional pipelines, trees and DAGs to arbitrary graphs. Determining the tradeoffs between the required resource count (such as number of links per cell with associated control logic) and the ability to emulate a broad range of structures required to represent data and/or control flow relationships is one of the subjects of the ongoing research.

3. Parallel Control Flow

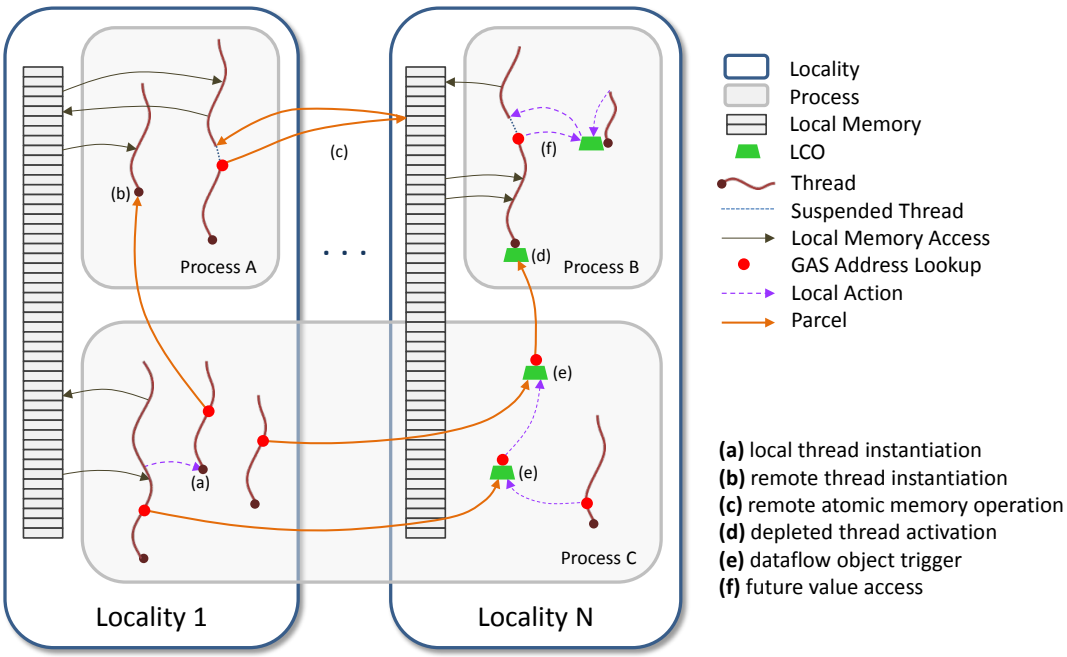


Figure 3. Semantic constructs of ParalleX enabling various types of parallel control flow

The CCA/Simultac architecture described thus far satisfies the need to define organization of primitive components. But the von Neumann architecture also provides the semantics of computing; the execution model that governs the logic of operations and the meaning of the

objects on which they perform as well as the order in which they are performed. Continuum Computer Architectures in general and the Simultac architectures in particular also require an execution model to transform the local rules and state to global general purpose computation, meeting the criteria of success required of this non von Neumann approach. The experimental ParalleX execution model serves this purpose as an abstraction to define and guide the semantics of computing implemented as a version of the HPX family of runtime systems [18, 19]. This also provides a low-level application programming interface (API) or a target for high-level language compilers. There are many aspects of the ParalleX execution model and its HPX embodiment that distinguish it from the classical von Neumann model, but a few functional constructs create the framework for the rest of the parallel operation. These include the following:

- Global name space – data, control, and program objects all exist in a single but hierarchical name space that permits virtual objects to migrate across physical space without requiring name change.
- Spanning processes – first class ephemeral contexts that include data, control state, task instantiations, child processes, and resource mapping that span multiple physical work units like conventional SMP nodes. They define the hierarchical name space.
- Compute complexes – the principal form of executing instantiated tasks that operate within a defined locality (contiguous physical space of bounded response time and guaranteed compound atomic sets of operations) that performs the work of the application.
- Parcel-driven computing – a form of active messages that move work to the data as opposed to always moving data to the work to reduce latencies and their effects while managing asynchronous operation of distributed systems.
- Local control objects – small objects that contain and transform control state a graph of which provides global parallelism control and continuations. Rich semantic constructs like dataflow and futures (actor's model [9]) provide a rich array of asynchronous control for managing the progress of complexes and the creation of new ones.

The HPX runtime systems manage the overall resource management and task scheduling of a ParalleX program anticipated for future CCA and Simultac application programs. HPX-5 is a recent experimental runtime system embodying many of these policies on conventional parallel computers for many applications in use today. Challenges in the area of overhead of runtime control can be minimized by architectural features within the fontons. This represents future research.

4. ExaOPS System Architecture

The proposed architecture may be implemented using the currently available CMOS technology (fig. 4). To lower the production costs, die sizes that result in best balance between the effective silicon area, yield, and ability to support the required number of I/O leads are selected, typically in the vicinity of 100 mm^2. In many cases the yield may be improved further since chips containing a few damaged fontons need not be discarded. The practical fonton counts are expected to reach about 10,000 per die. The required packaging density may be achieved by stacking dies on top of each other and connecting them using Through-Silicon Vias [4] to provide high local interconnect bandwidth while sacrificing relatively minor fraction of usable silicon area. The bottom dies incorporate high-speed serial transceivers to enable communication be-

tween different stacks on the same board. For longer distances, integrated silicon photonics [17] may be deployed to allow the use of more efficient fiber optics.

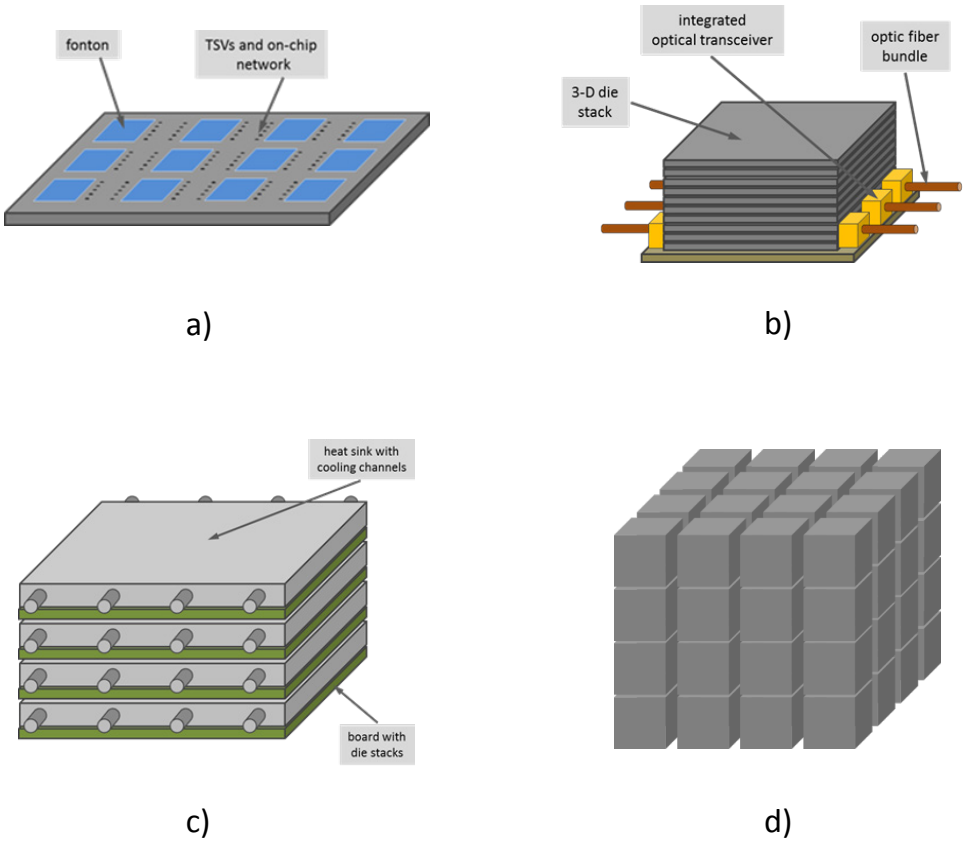
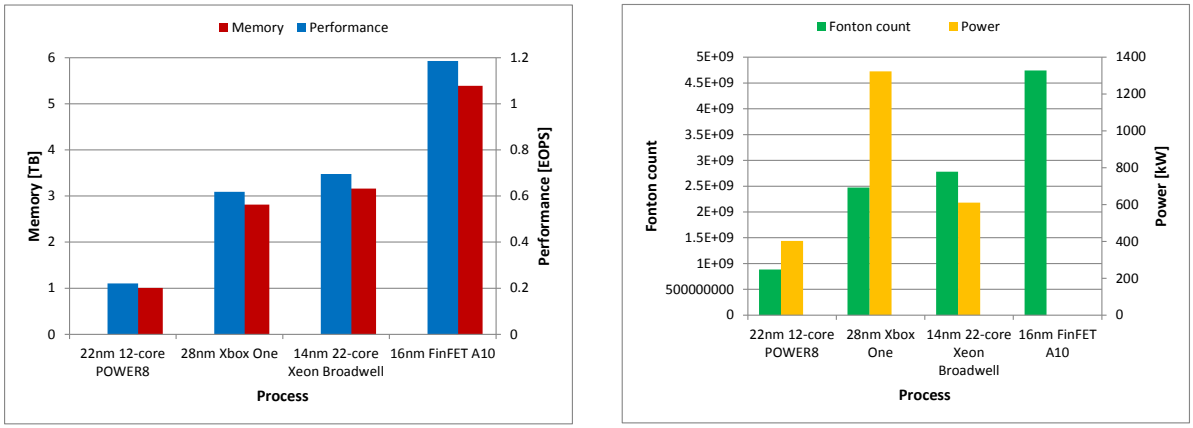


Figure 4. Simultac system architecture: a) single CMOS die, b) 3D die stack, c) board stack with integrated cooling, and d) exascale machine

The board space is mainly consumed by few thousands of die stacks. The remainder is occupied by voltage regulators and other conventional circuitry such as clock and reset generators, system monitors, etc. Relatively dense packaging will result in substantial heat dissipation thus preventing the use of conventional forced-air cooling. To cope with that, heat sinks with integrated cooling channels may be applied. This also permits the boards to be densely stacked as illustrated in fig. 4c. Finally, the complete system incorporates a number (64 shown) of board stacks arranged uniformly in three dimensions to minimize the length of cables connecting the individual units.

To estimate the resources needed to implement a Simultac system of exascale capability, a number of (rather conservative) assumptions have been made. To limit the power consumption, the clock frequency has been reduced to 250 MHz. The logic of a single fonton was estimated at the generous 16,000 gates, excluding memory. 50% of die resources are used for storage, while the logic and interconnect consume 40% and 10%, respectively. Each memory data cell uses 12 transistors per bit due to multi-ported access supporting both local operation and adjacency interfaces. The extra transistors may also be used in support circuits that improve noise margins in storage cells and reduce the current leakage arising as an effect of sub-1V supply voltage [13, 22], thus providing additional power saving. The assumed resource budget yields about 1 KB of useful storage per fonton, accounting for address tags and synchronization bits for individual words. The dies are stacked four-deep and the resulting stacks are spaced



a) b)

Figure 5. Comparison of one cubic meter Simultac implementation using four process technologies: a) overall performance and aggregate memory size and b) total fonton count and power dissipation. The estimated power consumption for the last technology node could not be calculated due to unavailability of reliable data

20 mm center-to-center on the boards, leaving 10% of board area for supporting circuitry. With boards mounted every 20 mm in vertical dimension, the properties of a prototype fitting into $1\,m^3$ are plotted in fig. 5 assuming transistor densities represented by four recent processor implementations: IBM POWER8, AMD APU for Xbox One, Intel Xeon Broadwell, and Apple A10. The power figures were approximated by linear scaling of the published TDP values by clock frequency and transistor count, hence do not take correctly into account static power dissipation and possible additional savings due to reduction of supply voltage. Properties of a full-scale system using arrangement of $4 \times 4 \times 4$ cubes are contrasted in tab. 1 with the currently fastest machine on the planet, TaihuLight. While Simultac compares favorably on nearly all metrics, it has significantly worse storage capacity. This may be corrected by integrating additional DRAM dies into 3D stacks and connecting them by TSVs to achieve high data throughput.

Table 1. Comparison of major system metrics for Simultac and TaihuLight

Parameter	Simultac	TaihuLight
Clock speed	250 MHz	1.45 GHz
Processing unit count	303.6 billion fontons	83.9 million FPUs
Peak performance	76 ExaOPS	125 PetaFLOPS
Total memory	345 TB	1.28 PB
Aggregate memory bandwidth	1821 EB/s	5.46 EB/s
Memory size to performance ratio	0.0000044 bytes/OPS	0.01 bytes/FLOPS
Memory bandwidth to performance ratio	24 bytes/OP	0.044 bytes/FLOP
Physical footprint	$25\,m^2$	$605\,m^2$

Conclusions

This paper has described a genus of computer architectures referred to here as the "Continuum Computer Architecture" and a particular specific instance, the "Simultac", intended to deliver superior performance to cost in the trans-exaOPS performance regime. It has taken

a distinctly different direction from typical incremental changes to conventional practice most frequently pursued. In particular, CCA is a genus of architectures that are non von Neumann in form and function for the purpose of eliminating the legacy artifacts of prior art that impose potentially severe bottlenecks. These include sequential instruction issue, separation of processing logic and memory often referred to as the von Neumann bottleneck, and the emphasis on FPU utilization for which much of the remaining architecture is dedicated. With the end of Moore's Law at nanoscale feature size, the development of revolutionary architectures suggests the most promising approach to dramatic improvements to efficiency and scalability. This paper has discussed at length the strategy of the CCA and has analyzed to first order the potential peak capability of the Simultac with respect to foot print and volume.

There are many design decisions yet to be determined for the Simultac class of cellular structures. Perhaps most significant is the granularity of the fonton components. How large and what is the form factor of the fonton storage? What are the logical functions that are directly implemented in hardware of the fonton both to perform the application operations and to eliminate sources of overhead for resource management and task scheduling? One parameter to be determined is the clock rate which needs to be slow enough to permit single cycle operation of the fonton but fast enough to maximize the sustained performance of the Simultac. The communication protocol has yet to be devised that can efficiently represent the Parcel semantics while minimizing latency and power. Managing the global name space and converting the virtual address space distribution to physical routing algorithms requires refinement with emphasis on race conditions due to asynchrony in the presence of migrating objects including continuations. An exciting opportunity for academic research in architecture is the design of the fonton. Conventional processor architectures are outside the scope of small teams of researchers. But this particular class of architecture, that is the CCA genus, yields itself to small groups of developers. Prototyping with FPGAs is entirely feasible that can lead to very small chips for proof of concept in semiconductor technology. The HPX runtime system software has been developed in multiple versions and is sufficiently mature to anticipate its use in this context. But changes to the low-level backend interface will have to be produced to work with the metal. This is a very exciting time in computer architecture beyond an exascale and at the end of Moore's Law. No longer should research be constrained by incrementalism of conventional practices. The need is too great to reconsider the opportunities of the non von Neumann family of architectures. An exaOPS in a cubic meter is an extraordinary goal, but realistic in its execution.

References

1. Anderson, M., Brodowicz, M., Kaiser, H., Sterling, T.L.: An application driven analysis of the ParalleX execution model. CoRR (2011), `http://arxiv.org/abs/1109.5201`, arXiv:1109.5201v1

2. Argyris, J.H., et al.: Finite element method – the natural approach. Computer Methods in Applied Mechanics and Engineering 17–18, 1–106 (January 1979), DOI:10.1016/0045-7825(79)90083-5,

3. Berlekamp, E.R., Conway, J.H., Guy, R.K.: Winning Ways for your Mathematical Plays, vol. 4. A. K. Peters Ltd. (2001-2004), ISBN:978-1568811444

4. Black, B., et al.: Die stacking (3D) microarchitecture. In: Proceedings of the 39^{th} Annual IEEE/ACM International Symposium on Microarchitecture, MICRO'06. pp. 469–479 (December 2006)

5. Borkar, S., et al.: Supporting systolic and memory communication in iWarp. In: Proceedings of the 17^{th} Annual International Symposium on Computer Architecture. pp. 70–81 (1990), DOI:10.1109/ISCA.1990.134510

6. Cooley, J.W., Tukey, J.W.: An algorithm for the machine calculation of complex Fourier series. Math. Comput. 19, 297–301 (1965), DOI:10.2307/2003354,

7. Dennard, R.H., Gaensslen, F., Yu, H.N., Rideout, L., Bassous, E., LeBlanc, A.: Design of ion-implanted MOSFET's with very small physical dimensions. IEEE Journal of Solid State Circuits 9(5) (October 1974), DOI:10.1109/JSSC.1974.1050511,

8. Dennis, J.B.: Data flow supercomputers. Computer 13(11), 48–56 (1980), DOI:10.1109/MC.1980.1653418

9. Hewitt, C., Baker, H.G.: Actors and continuous functionals. Tech. rep., Cambridge, MA, USA (1978)

10. Intel Corp.: Intel Threading Building Blocks (Intel TBB) (2017), website, `http://www.threadingbuildingblocks.org`

11. Kaiser, H., Brodowicz, M., Sterling, T.: ParalleX: An Advanced Parallel Execution Model for Scaling-Impaired Applications. In: Parallel Processing Workshops. pp. 394–401. IEEE Computer Society (2009), DOI:10.1109/ICPPW.2009.14

12. Kale, L.V., Krishnan., S.: Charm++: Parallel programming with message-driven objects. In: Wilson, G.V., Lu, P. (eds.) Parallel Programming using C++, pp. 175–213. MIT Press (1996), ISBN:9780262731188

13. Kim, T.H., Liu, J., Keane, J., Kim, C.H.: A high-density subthreshold SRAM with data-independent bitline leakage and virtual ground replica scheme. In: IEEE International Solid State Circuits Conference. pp. 330–331,606. IEEE (2007), DOI:10.1109/ISSCC.2007.373428

14. von Neumann, J.: Collected Works, vol. 5, pp. 288–326. Oxford: Pergamon Press (1961), ISBN:0080095666

15. Slaughter, E., Lee, W., Jia, Z., Warszawski, T., Aiken, A., McCormick, P., Ferenbaugh, C., Gutierrez, S., Davis, K., Shipman, G., Watkins, N., Bauer, M., Treichler, S.: Legion programming system (Feb 2017), version 16.10.0, `http://legion.stanford.edu/`

16. Sterling, T., Kogler, D., Anderson, M., Brodowicz, M.: SLOWER: A performance model for exascale computing. Supercomputing Frontiers and Innovations 1(2) (2014), DOI: 10.14529/jsfi140203

17. Syrbu, A., Mereuta, A., Iakovlev, V., Caliman, A., Royo, P., Kapon, E.: 10 Gbps VCSELs with high single mode output in 1310 nm and 1550 nm wavelength bands. In: Proceedings of the Optical Fiber Communication/National Fiber Optic Engineers Conference. pp. 1–3 (February 2008), DOI:10.1109/OFC.2008.4528529,

18. The Center for Research in Extreme Scale Technologies: HPX-5 (Nov 2016), version 4.0.0 http://hpx.crest.iu.edu/

19. The Ste||ar group: HPX (July 2016), version 0.9.99, http://stellar.cct.lsu.edu/

20. Tim, M., Romain, C.: OCR, the open community runtime interface (March 2016), version 1.1.0, https://xstack.exascale-tech.com/git/public?p=ocr.git;a=blob;f=ocr/spec/ocr-1.1.0.pdf

21. Valiant, L.G.: A bridging model for parallel computation. Comm. ACM 33(8), 103–111 (1990), DOI:10.1145/79173.79181

22. Verma, N., Chandrakasan, A.P.: A 256 kb 65 nm 8T subthreshold SRAM employing sense-amplifier redundancy. In: IEEE Journal of Solid-State Circuits. pp. 141–149. IEEE (2008), DOI:10.1109/JSSC.2007.908005

23. Wilke, J., Hollman, D., Slattengren, N., Lifflander, J., Kolla, H., Rizzi, F., Teranishi, K., Bennett, J.: DARMA 0.3.0-alpha specification (March 2016), version 0.3.0-alpha, SANDIA Report SAND2016-5397

DOI: 10.14529/jsfi170204

The Simultaneous Transmit And Receive (STAR) Message Protocol

Earle Jennings[1]

The STAR protocol is proposed, which solves three inherent problems with MPI, a well known security problem caused by data memory access faults, and the following four exascale communication problems. Exascale systems must efficiently save the state of their Data Processor Chips (DPCs) every fraction of a second to support rollback when an error is encountered. Exascale systems will have the possibility of frequent optical communications failures. Exascale systems must resiliently respond to these optical communication failures. An exascale computer needs to be fed enough data fast enough, and its data center must keep up. Optical implementations are developed compatible with 100 Gbit/sec Ethernet to solve these problems. Automatic fault resilience mechanisms are discussed, which improve HPC quality of service, and meet the exascale reliability and resilience challenges. The bandwidth problem for exascale computers interfacing with data centers is solved. The STAR protocol enables data centers and supercomputers, to be invulnerable to memory fault injection of viruses and rootkits.

Keywords: MPI, optical communications, exascale, HPC, security, router, memory wall, fault resilience.

Introduction: Problems to be Solved

MPI is a standard for message passing commonly found in parallel processing systems, in particular HPC systems [8], [17], [7],and [15]. While MPI is versatile and readily available, it has inherent problems. Figure 1 shows sending a message in MPI. The sending of the message locks up the buffer until the message has completed being sent. All instruction processing has to refrain from accessing the buffer until the buffer is unlocked. Often, the core must rely upon an interrupt structure to know when the buffer is again ready for access, implement a wait loop, or suspend a thread. This not only slows things down, but costs energy and increases complexity of the hardware and software operating environment.

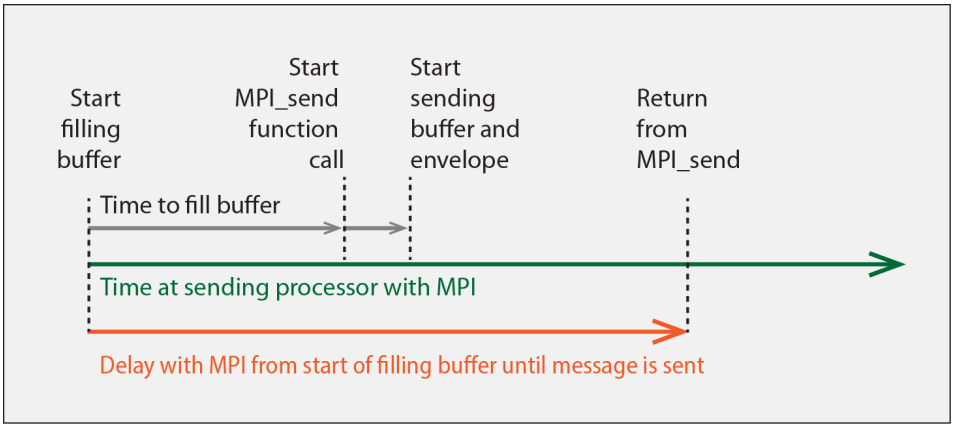


Figure 1. Sending a message in MPI

Figure 2 shows receiving the MPI message not only locks up a buffer until the message is received, but also for the time required to process, or move, its contents elsewhere. Making this function efficient now requires that the operating environment know when this buffer mechanism

[1]QSigma, Inc., Sunnyvale, USA

can be accessed for a new operation. The operating envirnoment may use one or more of the following: status registers to trigger an interrupt in the core; implement a second wait loop; or suspend the thread. From an application program perspective, these operations appear to be primitives, but they come with a large amount of overhead, in terms of instruction processing to initiate these operations, wait for their completion and then release these resources to be used for a new operation. All of this takes energy, hardware, and computing time.

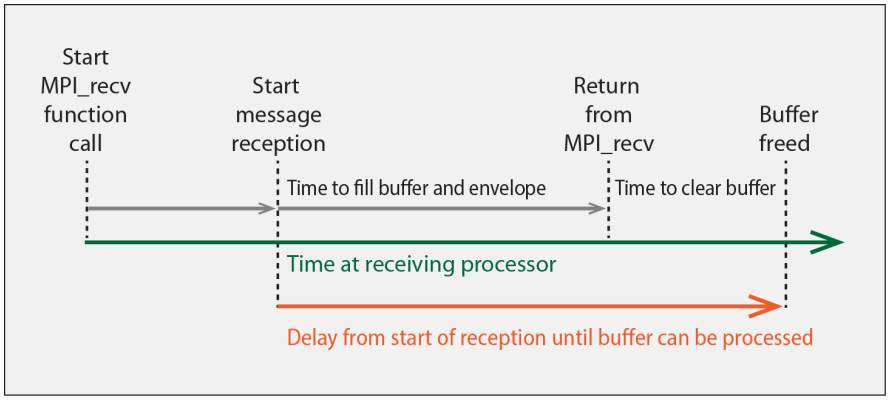


Figure 2. Receiving a message in MPI

With MPI, messages can be any length, and a short message can be stalled by a long message at a router transfer point, as shown in Figure 3. It should be noted that Intel has announced a component solution to this one problem, but it does not solve all three of these MPI related problems, much less the security, and the exascale related problems we consider next.

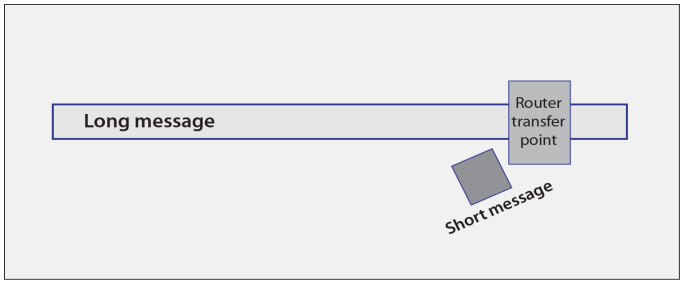


Figure 3. A short message stalled at a router transfer point by a long message

There is common security problem in processors derived from the PDP-11 and/or the IBM 360 [1]. A buffer is assumed to have a starting address, a length, and from this, a computed ending address. Suppose a computer starts writing at the starting address, but writes a string that is longer than the length of the buffer. In other situations, the writing goes toward the start of the buffer and writes data into addresses before the buffer's starting address. In either case, addresses outside of the buffer are altered. For example, these addresses can hold program instructions, pointers to code, such as the return address for a program during execution. These data buffer access faults can, and do, inject viruses or rootkits into the instruction memory of these processors [19]. Data fault injection turns seemingly innocent data into an injected virus, which can then access anything in the memory domain of the injected device. While this issue has been around for decades, it persists as a major security problem [13]. One of the earliest examples of this ocurred in 1988, when the Morris worm, infected DEC VAX and SUN computers running BSD Unix across the Internet [4]. This article shows how the STAR communications protocol supports rendering this situation impossible for a data center or a supercomputer.

Exascale brings with it four new problems.

1. Consider a Data Processor Chip (DPC) containing some increment of 20 Mbytes of internal state. An exascale computing system [3] will probably have at least several hundred thousand DPC's, each containing at least 500 cores. Further assume that for any specific fault, the system needs to be able to rollback back within a second, with an overhead of no more than a few percent, say 2%. Consider what is required when the DPC state is captured in a snapshot every 10 ms. Within that 10 ms, 2% of the time the DPC is saving the state, with each DPC operating on a local 1 ns clock. (All mention of a clock only refers to a local clock.) This implies that each 20 Mbytes must be saved in 2% of 10M clock cycles, or 200K clocks. Therefore, for each 20 Mbytes of internal state in the DPC, 100 bytes must be saved in a nanosecond. This is much more than the 100 Gbits/second optical fibers can support. While rolling back by a second seems natural, the question is how much of the system needs to roll back. By recording snapshots every 10 milliseconds, in many situations, the amount of roll back can be limited to a much smaller fraction of the system, saving energy and computing capacity.
2. Another problem is the failure rate for large scale optical networks. These failures can be defined as any message that is received, and through analysis of its Error Detection/Correction (EDC) envelope, is found to have detectable, but uncorrectable, errors. Failure rate is the number of failures per unit time. The overall communication failure rate of the exascale system [3] depends upon the failure rate for one optical fiber and its transceivers, and grows directly (but not necessarily linearly) based upon each of the following: The number of data channels per bundle interfacing to each of the chips; the number of optical fibers in each channel; the number of DPCs; the number of routers needed to avoid deadlocking these chips; and the complexity of the routers.
3. Analyses [3] of the maintenance logs of current US supercomputers indicate that the Mean Time Between Failure (MTBF) for exascale systems may be no more than minutes. When the optical communications fail, the system fails. With exascale systems, there is no time to rely upon an operating system, or a distant snapshot of the communications. This led the author to develop this multiple degree of freedom, fault resilience strategy, to be implemented in the local communications hardware.
4. There is a another, essentially mirror problem to the first exascale problem. How does one feed data into an exascale computer fast enough to keep it busy? This problem will be answered later at three system component levels, first at the DPC, second at the mostly binary topological network interfaces of the system components, and third at the cabinet to data center interface level.

1. Introducing the STAR Protocol

It is required for STAR message protocol compliance that any compliant device transmits and receives a STAR message on every local clock cycle, except when responding to an uncorrectable error upon reception. This requirement removes the three problems with MPI. Sending a message is completed in one local clock. Receiving a message is completed and buffer released in one local clock. Each STAR message clears each local pipe stage in the routers in one local clock, so that no message stalls another for an unknown amount of time. The response to such errors can involve automatic channel component replacement in, at most, a microsecond.

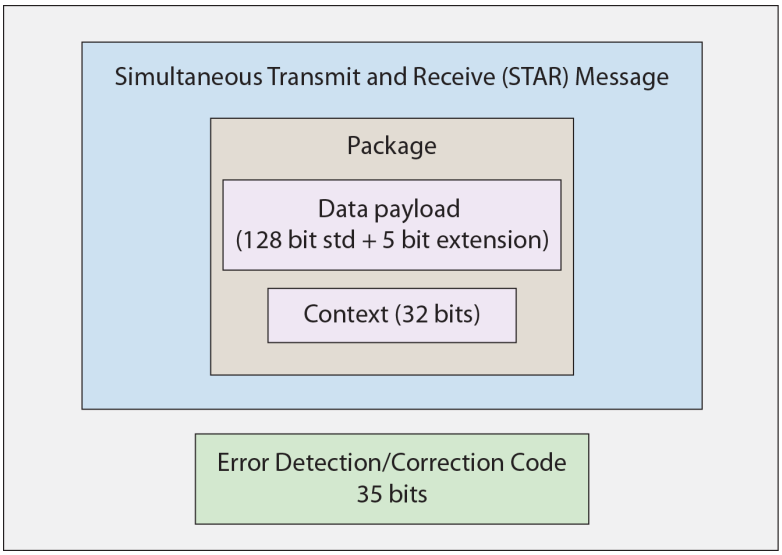


Figure 4. STAR message transport layer

Figure 4 shows the application layer of the STAR message, which includes a package containing a data payload and a context. The payload includes a 5 bit extension code and 128 bits of standard payload. The context is 32 bits, which is interpreted at every STAR message core to determine the package disposition and transfer. The context and its interpretation, is under complete control of the program.

The transport layer of each STAR message, also includes a 35 bit Error Detection/Correction (EDC) code. These 35 bits are arranged in clusters of 7 bits. Each 7 bit cluster relates to a separate group of 33 bits of the package. There are five groups covering the package. These 7 EDC bits allow the system, at each receiver, to calculate a 1 bit error correction of the received group of 33 bits. These same 7 EDC bits enable calculation of a 2 bit error detection flag for these same 33 bits of the received package. All of these EDC calculations are required to be performed within one, or more, locally clocked (nanosecond) pipe stages. The transport layer supports detecting and correcting up to five single bit errors, one in each of the groups of the application layer. The transport layer also supports detecting any, or all, of the five groups having uncorrectable 2 bit errors. The local clock is about 1 GHz, and is readily supported by standard CMOS, so that 200 Gbits/second can be simultaneously sent, and received, on each STAR channel.

1.1. Exascale State Resilience and Optimized Matrix Communication

The first example of the payload format, sets the top Extension bit (Ext bit 4) to 0, and the remaining payload is interpreted as two double precision floating point numbers, each augmented by two bits of the extension field, used as guard bits. This format can represent a complex number, as well as be useful in bulk downloading and uploading of the state of cores in the DPC. Note that each of these messages is transferring 16 bytes in one clock cycle, so that 8 of these channels delivers 128 bytes per nanosecond in and out. Recall that each DPC has some multiple of 20 Mbytes of internal state, which required 100 bytes per nanosecond to store, and restore, as discussed above with a reasonable overhead for application programs. This approach makes DPC state resilience feasible. The STAR bandwidth delivery, both as input and output,

is a necessity for keeping an exascale system fed with data, which is further discussed in the following section on the optical implementation below.

The second format example configures Ext bit 4 as 1, and two of the other 4 bits as 0. This can be interpreted as a double precision number and its two guard bits with an index list of 64 bits. This format has two uses. In dense matrix manipulation, such as matrix inversion by Gaussian elimination with partial pivoting, or block LU decomposition [6], the format is used to communicate a pivot, or potential pivot candidate. In sparse matrix manipulation [10], [16], 4 bits of the index list can act as a field identifying sixteen large objects, such as sparse matrix A and several vectors associated with the matrix. This format configuration insures efficiency for both dense, and sparse, matrix operations. By configuring the context of a message, the second format can be interpreted as having more than one floating point number. For example, the 66 bits can be interpreted as dual single precision (each with a guard bit), or quad FP16 bit numbers. Further configurations using the context, can result in these 66 bits being interpreted as numeric values in a form of posit numbers [9]. With posit numbers, more opportunities are potentially available, for instance, 3 posit numbers, each of 22 bits in length, or 5 posits, each of 13 bits, or 6 posits, each of 11 bits. The net result of these opportunities is that as an application progresses in run time, it can use fewer parameters per payload, for greater precision.

2. STAR Bundle

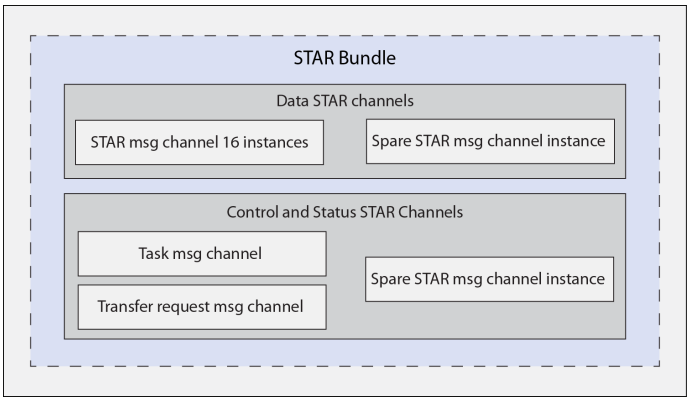


Figure 5. Example of a STAR bundle

Figure 5 shows a STAR bundle including data channels as well as control and status channels. There are 16 data channels, with a spare channel instance used for fault resilient recovery from unrecoverable message faults. The control and status channels include one task and one transfer request message channel, and a spare channel for fault resilient recovery. This example has several positive consequences. 16 data channels guarantee that the state of the DPC can be saved within a reasonable time overhead for exascale systems. Note that in the above discussion, we showed that 8 channels deliver 128 bytes per local clock, so 16 channels deliver 256 bytes per clock. This is a data payload bandwidth of 2 Terabits per second input and output of each link (bundle interface) for every DPC, each memory controller and every link of every router in the system. Every DPC can receive and send 32 double precision numbers on every local clock. This is compared to the Sunway [2], with a system interface at the MPE/CPE chip of 16 Gbytes/second = 128 Gbits/sec with a latency of 1 microsecond. The ratio of 2 Tbits/256 Gbits is roughly 16. The Sunway is roughly 93 Petaflops of performance, or 9% of an exaflop. Delivering 16 times the bandwidth for 11 times the performance has a reasonable expectation of reaching exascale

system performance. The transfer request messages can remove all the main memory address calculations from the DPCs, sending instead just the necessary parameters for the transfers. This results in energy reduction for many programs in the DPC, by removing the address calculations from the DPC, and placing them in the memory controllers. Task messaging is carried out in a separate channel from data transfers and their requests, systematically separating data from task control and configuration. Therefore, data cannot be confused with task control and configuration infromation by these separate channels. This removes an avenue of injection for malicious software such as viruses and root kits.

2.1. STAR Communications Network

A STAR communications network is a point-to-point network of nodes, each acting as sources, destinations, and routing nodes, called STAR Trinary Routers (STRs).

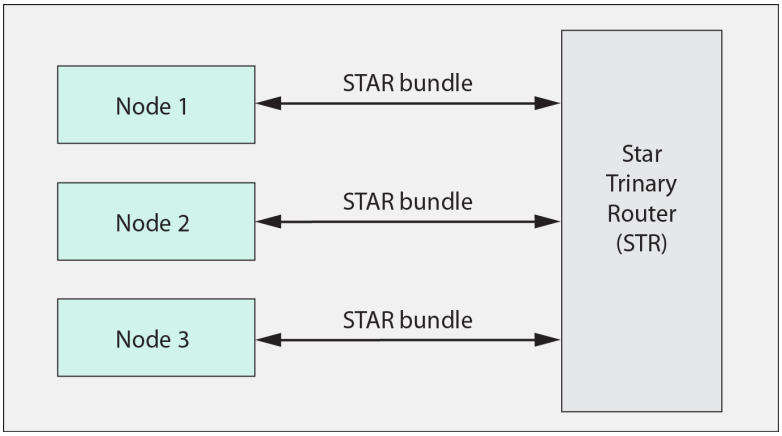


Figure 6. STAR Trinary Router

Each STR has three bidirectional STAR bundle links to nodes, or other STR instances, as shown in Figure 6. Each node may be a module within a chip, or a chip. The interface between a STAR bundle and data processing circuitry, whether in an STR, or a core module, is through a STAR bundle module.

Figure 7 shows a STAR in-chip network, including a binary graph of channel bundles, whose nodes are interfacing through bundle modules to a core, or core modules, called the Programmable Execution Modules (PEMs) 0 to 3. The STRs inside a chip are laid out as a communication module paired with a module of cores as a unit. In particular, the PEM and the STAR bundle module with the STR are laid out as a unit.

Figure 8, on the left side, shows a STAR channel core as part of STAR bundle module 1. The data STAR channel core interfaces a data channel of the STAR channel bundle shown in the middle. This interface includes an Incoming Message Processor (IMP) and an Outgoing Message Processor (OMP). The IMP and the OMP belong to the data STAR channel core 1. They interface with a core, possibly in a core module, or in the PEM, or in the STR. On the right side, each of the STAR channels of the second STAR bundle module interface through a corresponding channel core to another core, core module, PEM, or STR, either within a chip, or between chips.

Consider block LU decomposition [6] on a matrix with 32 million rows and columns, which is the approximate size required by the Linpack specification for a computer to be exascale. Assume the block size is 32 rows and columns. This matrix will be processed for eight or more

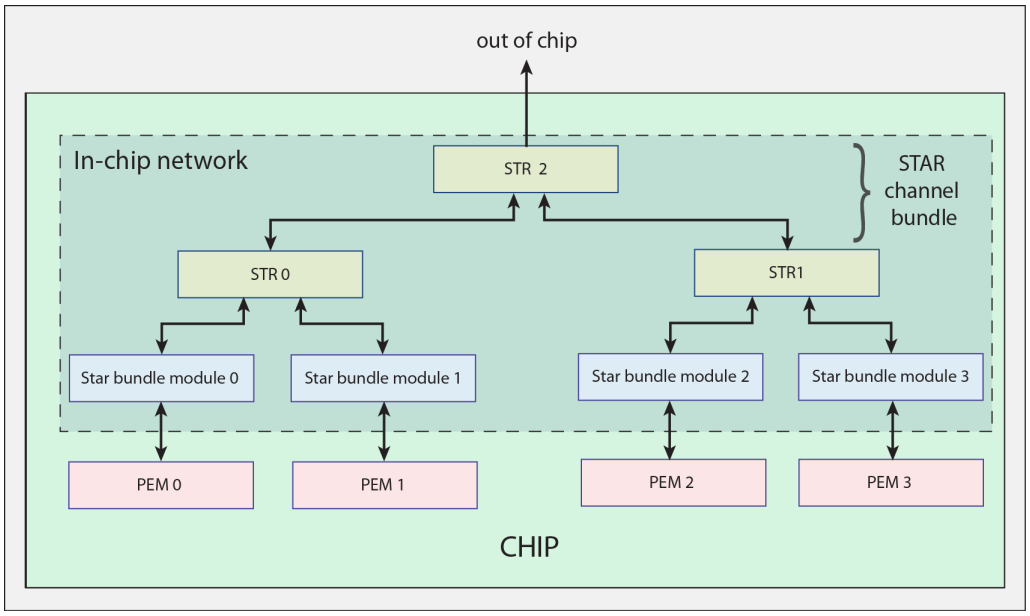


Figure 7. A STAR in-chip network

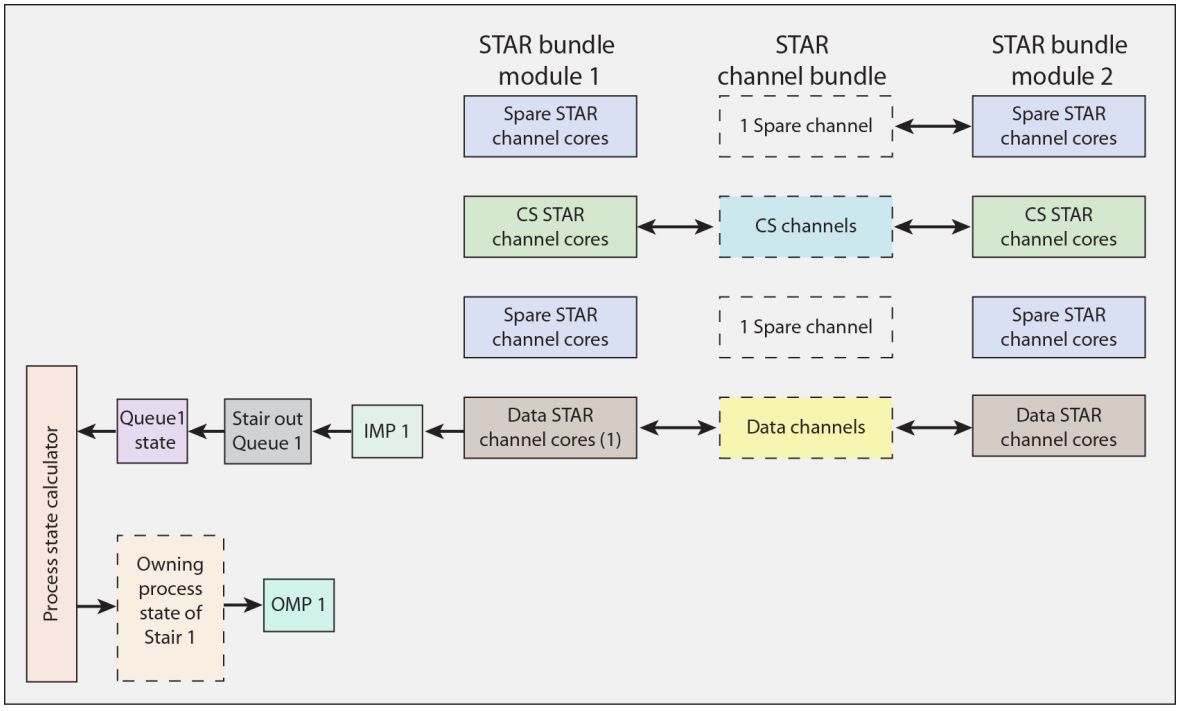


Figure 8. Interfacing a STAR channel bundle between two bundle modules

hours to meet the HPL standard of sustained performance of an exaflop [14]. At the start, the probability of row swapping is essentialy certain, as there is one chance in 32 million the pivot is in the diagonal entry. This persists for most almost all of the runtime of the algorithm, say for 32 million minus 32 of the diagonal entries. When processing of the next block column begins, that block columns needs to be loaded, which means that almost all of the rows have been swapped. This can be visualized as a swarm of insects all taking off and landing somewhere else in a random pattern. These swarms of messages must be efficiently delivered. For this to happen, each message needs to clear its inhabited pipe stage in one local clock, or the communication clogs, and the supercomputer will fail to perform Linpack at an exaflop. Also, if even one of these messages fails to be delivered, the Linpack benchmark may fail.

2.2. Optical Implementation of a STAR Channel

Communication in exascale systems requires optical communications for, at least, the following reasons.

- Assume a system includes the DPC chips arranged in chip stacks, on some form of Printed Circuit Board (PCB) interfaced through some form of motherboard to the rest of the system. Assume that the DPC states are stored in DRAM local to the chip stack so that the 20 Mbytes travels no more than 10 cm to its backup store. Otherwise communications on the PCBs is likely to clog.
- Delivering 800 bits per nanosecond into or out of one chip cannot be reasonably done with our classic level-shifted electronic signaling. Such signaling is best represented by LVDS (Low Voltage Differential Signal) protocols [5] for integrated circuits, which have a typical upper limit of about 1 bit per nanosecond per coupling. These couplings typically do not support bidirectional simultaneous signaling.
- 800 bits per clock requires at least 1600 pins, in each direction. Even in a ball grid array for a chip 5 cm on a side, this will saturate the active logic pins.

Electronically traversing 10 cm across 800 signal pairs is an RF noise nightmare with today's PCB technologies. While the basic PCB layout rules of thumb say that these 800 differential signal pairs are traveling at 1/2 the speed of light (30 cm per nanosecond) the quarter wave length of a 1 Ghz signal is 15 cm. However, the rising and falling edges of the differential signal pair have a large high frequency component of roughly 10% of the wavelength of the overall signal, or about 10 GHz. This means that anything more than about 1.5 cm can act as an antenna for the differential edges, so many of these signal paths can cross couple due to the antenna affect [20]. These very wide signal paths are assumed to be synchonized. Thirdly, consider the sheer number of interfaces. The roughly 500 K to a million DPC tells us, based upon the other problems just outlined, that the electromagnetic couplings of the past are not reliable enough for this function.

The above shows that communication between these DPC, their memory controllers in the chip stacks, and nodes connecting these chip stacks, PCBs and motherboards need to be optical in their transport of the data. The 100 bytes (800 bits) per nanosecond clock is 8 times the maximum rate of 100 Gbit optical Ethernet. This bandwidth must be achieved with a multi-fiber protocol approach. Consider multi-fiber links between these components in a HPC system sustainably generating 10^{18} floating point operations per second (flops). These links are going to require Error Detection and Correction (EDC) circuitry at each receiver for each end of each optical fiber. These EDC circuits are going to need to respond with good, corrected or

uncorrectable error detection status very quickly. Each transmitter and receiver side circuit is going to have to fix itself locally, save and restore the communication stream disrupted by the detected error, and keep track of what was sent until it is certain that it was correctly received. Again, all of this needs to be local to its side of the link, or else the system can clog. Consider the individual messages communicated on a STAR message channel.

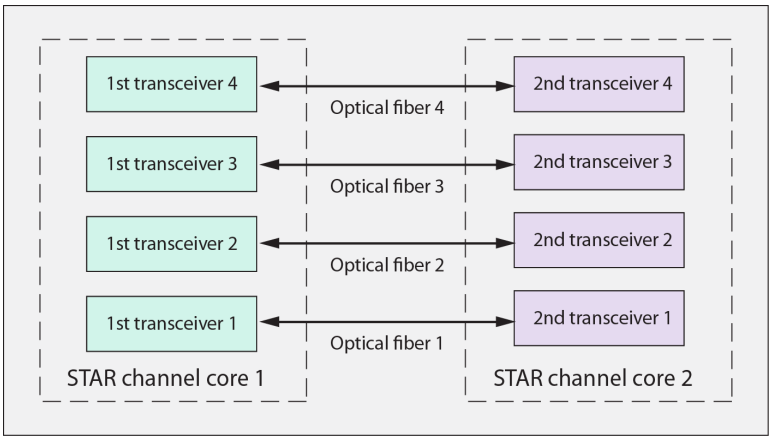


Figure 9. Example of fault resilient STAR optical implementation

Figure 9 shows an implementation consistent with today's 100 Gbit Ethernet opto-transceivers, using four optical fibers and associated transceivers, between two STAR channel cores. The STAR channel cores each interface to this single STAR message channel, which supports four degrees of freedom in automated, local, fault resilient response to an uncorrectable received message fault. In the best of situations, during normal operations, two transceiver pairs can be operated locally at 130-140 GHz for 100 Gbit/sec transfers across two of the optical fibers, delivering a STAR message every nanosecond to collectively deliver 200 Gbits per second across the two fibers as the first degree of freedom in the fault resilience strategy. In other situations, three transceiver pairs are operated locally at 100 GHz, for 66-70 Gbit/sec to collectively transfer 200 Gbits/sec across three optical fibers delivering a STAR message every nanosecond, as shown in Figure 10 as the second degree of freedom. Consider implementations where each STAR message is delivered by one optical fiber and its corresponding circuitry. The local STAR channel core, either receiving a message with an unrecoverable error, or in transmitting that message, can directly determine which fiber, and circuitry, failed. This information can be used by both sending and receiving STAR channel cores to resiliently respond to the error.

TS k stands for the Training Sequence of STAR message k, fork=1:7. In communication protocols, a training sequence refers to a known sequence of channel states which are transmitted to "train" the synchronization circuitry, such as phase locked loops or delay locked loops. The training sequence is often incorporated into each message or burst, to account for timing drift between the transmitter and the receiver. Note that all four optical fibers in Figure 9, can be operated to deliver the STAR message on every local clock cycle as the third degree of freedom in the communication fault resilience strategy. This has been not been shown in detail, but operates much the same as in Figure 10. To summarize, when everything is working optimally in a STAR optical channel, two of the four optical fibers are operated to achieve the collective delivery of a STAR message every (local) nanosecond. When the pairs of fibers and components are not able to operate at 130-140Ghz, a component trio operates at a lower frequency as outlined in Figure 10. When the trios can no longer operate in this fashion, then all four optical fibers and their

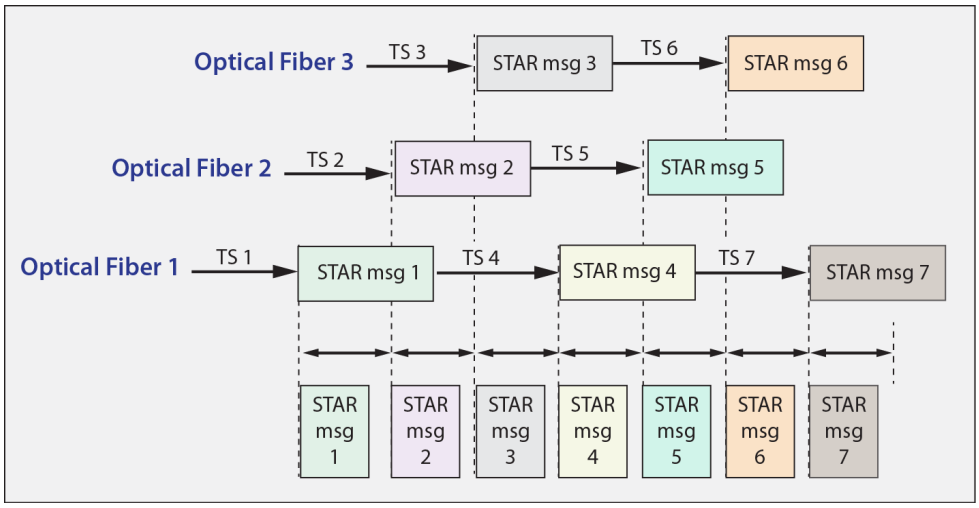


Figure 10. Transmitting and receiving STAR messages using three optical fibers

transceivers are operated at a second, lower frequency to collectively deliver the STAR message every nanosecond.

Figure 11 shows some of the details of the STAR channel cores supporting the interactions within this STAR channel. While there are four optical fibers being operated, each fiber supports communication in each of two channel directions, labeled channel directions one and two. During normal operations at the receiver in STAR channel core 1, ER1 in the upper left hand corner is not asserted, and the incoming selector selects the received message from channel direction one, shown along the left side. At the transmitter in STAR channel core 2 near the bottom, the outgoing selector stimulates the OMP 2. Consider the automated response when operating all four optical fibers fail in one direction. The faulty link, from transmitters to receivers in that one direction is automatically replaced by the spare optical fibers and circuitry in that direction.

Figure 12 shows the interactions in one channel direction, and in the spare channel direction, in greater detail. During normal operation, OMP 2 is stimulated with the package and the context from the outgoing context generator, unless the resend queue has a backlog. When the resend queue has a backlog, the package and context is sent from the queue directly, and any newly generated package and context is added to the existing resend queue. During normal operations the package and context are logged at the resend queue, in case this message fails to be received correctly. Assume the receiver's EDC pipe detects that the received message has an uncorrectable error. This indicates the received message is fatally flawed.

When the EDC pipe reports the uncorrectable error, ER 1 is asserted and a source error message is sent by one or more of the control and status channels from the destination to the source. When all four of the optical fibers for this message channel are already in use, the link treats both source and destination as tainted in this direction. Up until all four optical fibers are in use, the link will attempt repair by using either a different combination of optical fibers, or more optical fibers while operating at a slower local clock, at both the source and the destination. Assuming that all four optical fibers are in use, both the optical fibers, the source and the destination circuitry are replaced by the spare channel resources in the same direction. The operations, when completed, have replaced the left side of Figure 12 with the right hand side. This is the fourth degree of freedom in the communication fault resilience strategy.

Components, such as the DPC, typically have a binary tree interface for the STAR network as shown in Figure 7. Figure 13 shows a different level in a system, where binary trees are shown

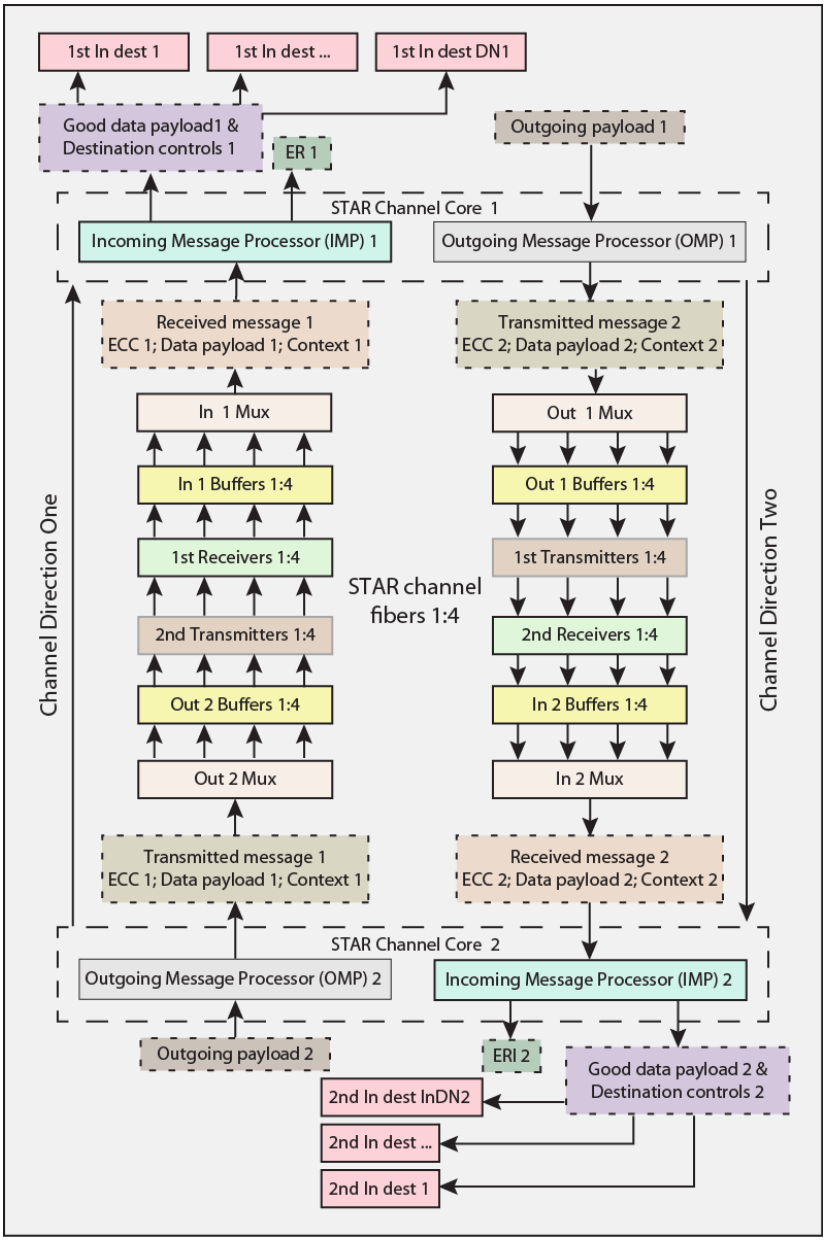


Figure 11. Interaction between two STAR channel cores across the four optical fibers

on the left as Link 1 and Link 2. If the whole system was composed entirely of these binary tree interfaces, the HPC components would be vulnerable to bottlenecks. Extending the network to a Mostly Binary Tree through the network and operation of STRs to form the PCB interface at Link3 and Link 4, enables optical PCBs etc., to have dual, or better, STAR bundle interfaces, eliminating the bottlenecks. This is another level in the understanding of how to keep an exascale computer fed with data.

Consider the communication between one of these supercomputers and its data center. The management requirement for such a communications interface is that it does not stall the supercomputer, and can keep up with its output. Figure 14 shows using the mostly binary STAR network to allow one cabinet to provide 15 or more STAR bundles to interface with the data center. Using these STAR bundles, each cabinet can use the 16 STAR data channels per bundle, each with 4 Ethernet optical fibers through the STAR bundle to ethernet nodes, as shown. This supports the cabinet simultaneously communicating with 500 or more Ethernet networks (15

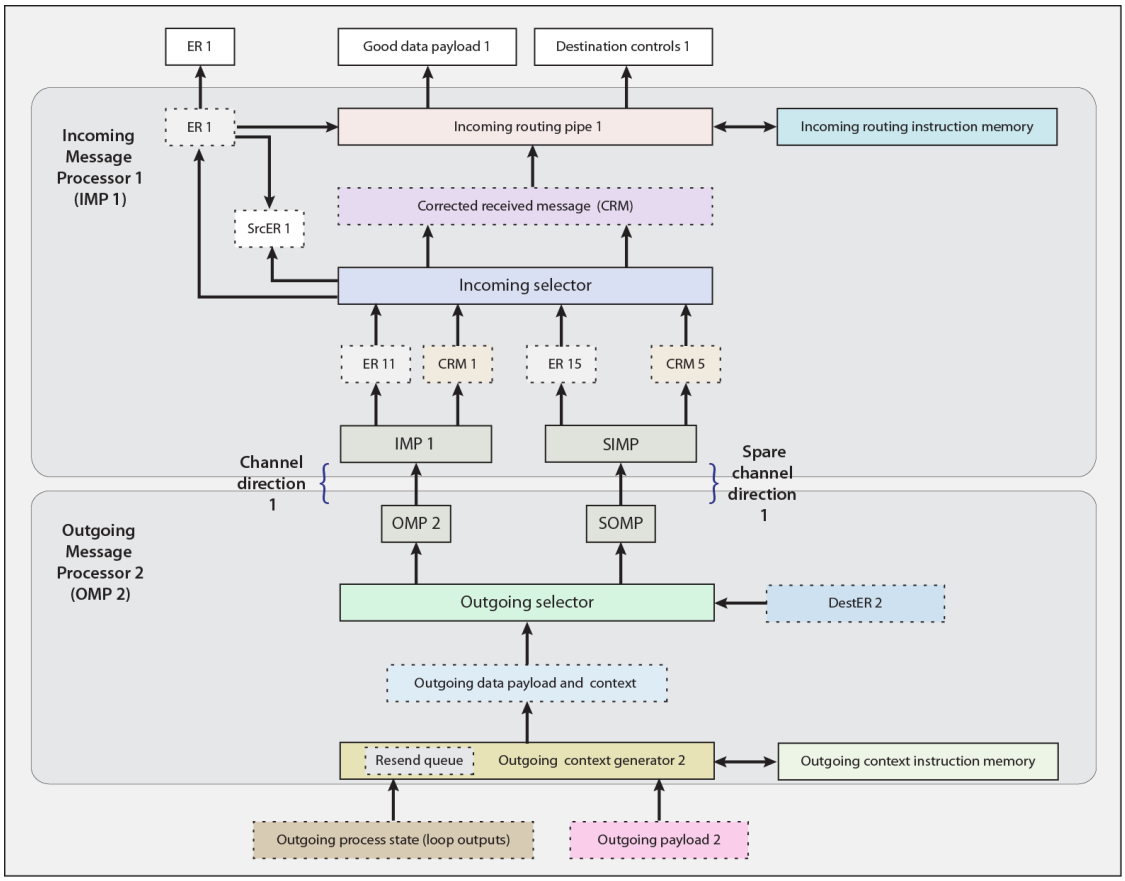


Figure 12. Detailed view of STAR channel core 1 interacting with STAR channel core 2

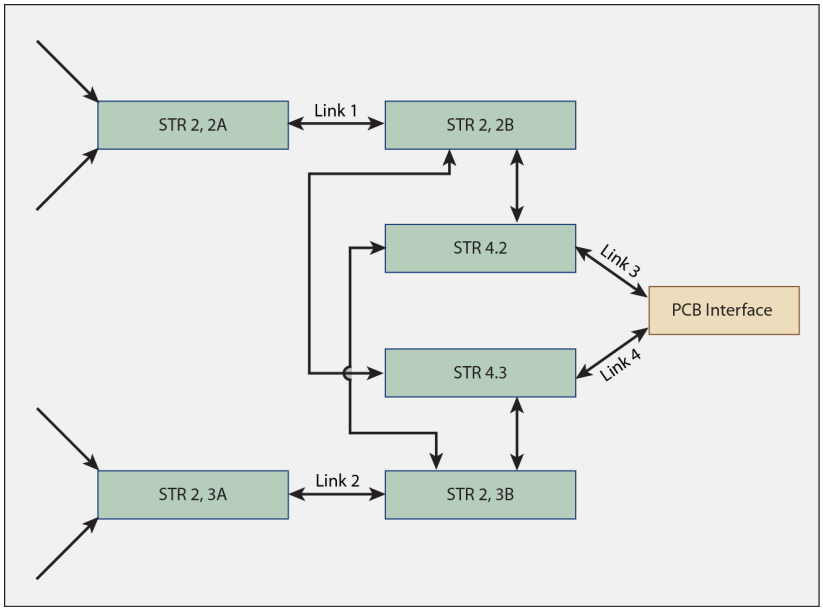


Figure 13. STAR binary tree components extended to make a mostly binary network component

bundles*16 channels/Bundle * 4 fibers/channel= 960), each with 100 Gbit/second bandwidth. This delivery resolves the communication bandwidth requirements to, and from, supercomputers for years to come. Note that this is the third level needed to understand how to keep an exascale computer fed with data.

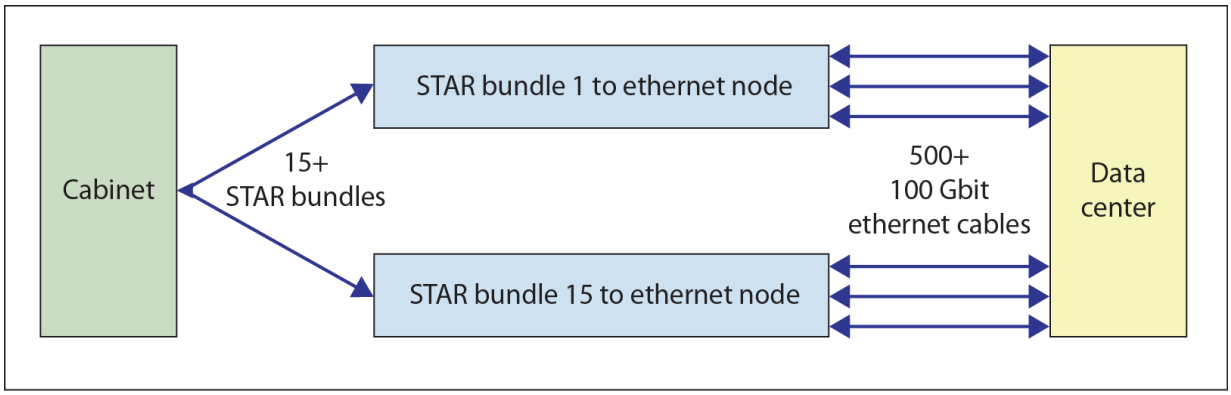


Figure 14. Cabinet to data center communications with the mostly binary STAR network

3. Transforming Today's Vulnerable Data Center

Today's data centers need to interact with something like the world wide web or the internet, let's call it a merged data and task-instruction external stream shown in the top right corner in Figure 15. The interactions can be regimented so that data communications go through an external data portal and the task-instruction related communications go through an external task portal. Further, one portal's communications cannot be intercepted by the other portal. There are a number of options for achieving this, such as different physical transport layers, differing communications protocols, and distinct encryption mechanisms, any, and all, of which can be implemented.

Second, the data center's internal communications are systematically segregated into one, or more, data related networks and task-instruction related networks. Each of these segregated networks has its own memory controllers. Each of these memory controllers only handles one of the two segregated information types. The data memory controller communicates between the internal data network and the various data related drives, DRAM and SRAMs. Also, the internal data network communicates only across data related channels to invulnerable handheld devices and invulnerable networked sensors [12]. The task-instruction memory controller only communicates between the internal task network and the task related drives, DRAMs, and SRAMs. Also, the internal task network communicates only across task-instruction information channels with the invulnerable handheld sensor and invulnerable handheld devices . Third, the cores, core modules, PEMs, or chips, separately communicate with the task-information messages, and the data-related messages, delivered by these segregated internal networks. These cores, core modules, etc. cannot alter their instruction memory nor their task control based upon data memory comminucation or access. With this approach, the data center of the future can be constructed to systematically exclude the possibility of data memory faults triggering the injection of viruses and root kits.

4. Summary

The STAR message protocol solves, or enables the solution of, each of the following problems:

- Sending an MPI message locks up a buffer until the message has completed being sent.
- Receiving an MPI message locks up the buffer until the message is received and it has either been processed, or cleared from the buffer.
- Short messages can be stalled by long messages at a router transfer point in MPI.

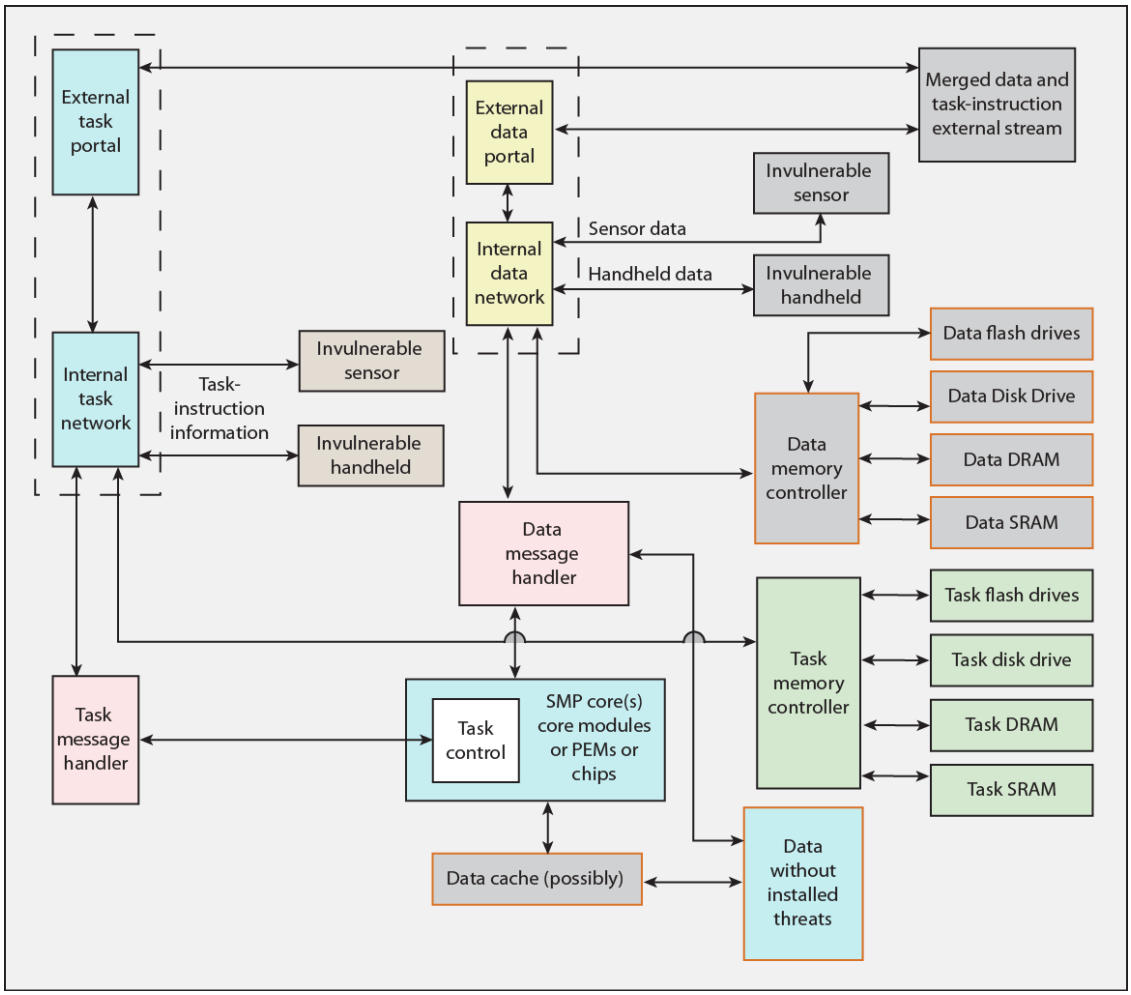


Figure 15. Tomorrow's invulnerable data center

- The common data memory fault security problem, which can inject viruses and rootkits into a computer system.
- The bandwidth problem involved in saving the state of DPCs in exascale systems.
- The data center problem of feeding enough data into, and out of, an exascale system so the exascale system is not stalled.
- The failure rate problem for optical communications in exascale systems.
- The fault resilient response to uncorrectable errors in a received optical message.

References

1. Chisnall, D.; Rothwell, C.; Watson, R. N. M.; Woodruff, J.; Vadera, M.; Moore, S. W.; Roe, M. Davis, B.; Neumann, P.G. (March 2015) "Beyond the PDP-11: Architectural support for a memory-safe C abstract machine", ASPLOS '15, March 14–18, 2015, Istanbul, Turkey, ACM 978-1-4503-2835-7/15/03, DOI: 10.1145/2694344.2694367

2. Dongarra; Report on the Sunway TaihuLight System; (2016), University of Tennessee, Oak Ridge National Laboratory, Dept Electrical Engineering and Computer Science, Tech Report, UT-EECS-16-742, US

3. DOE-ASCAC Subcomittee Report, Top Ten Exascale Researcgh Challenges, (2014 Feb 10), USA

4. Dresler; "United States v Morris", Cases and Materials on Criminal Law. St. Paul, MN: Thomson/West. ISBN 978-0-314-17719-3.

5. "AN-5017 LVDS Fundamentals", (2005) https://www.fairchildsemi.com/application-notes/AN/AN-5017.pdf

6. Golub, van Loan; Matrix Computations (4th ed); (2013) Johns Hopkins University Press, Kindle Edition, Baltimore, Maryland, US

7. Gropp, Huss-Lederman, Lumsdaine, Lusk, Nitzberg, Saphir, Snir; MPI-The Complete Reference vol 2, The MPI extensions, (1998) MIT Press, Cambridge, Mass., US

8. Gropp, Lusk, Skjellum; Using MPI: Portable, Parallel Programming with the Messaging Passing Interface (3rd ed), (2014) MIT Press, Cambridge, Mass. US, DOI: 10.1524/9783486841008

9. Gustafson; Beating Floating Point at its Own Game: Posit Arithmetic; http://supercomputingfrontiers.com/2017/wp-content/uploads/2017/03/2_1100_John-Gustafson.pdf (2017)

10. Heroux, Dongara, Luszcek; HPCG Technical Specification, SAND 203-8752, www.osti.gov/scitech/biblio/1113870f (2013), Sandia National Labs, US

11. HPC Advisory Council, Introduction to High-Speed InfiniBand Interconnect, US

12. Jennings; (Dec 2016) "Simultaneous Multi-Processor Cores for Efficient Embedded Applications", delivered in Barcelona, Spain, and scheduled for publication in the Journal of Computing

13. The Mitre Corporation, (September 2011) "2011 CWE/SANS Top 25 Most Dangerous Software Errors" © 2011 The Mitre Corporation, http://cwe.mitre.org/top25/, downloaded Oct 1, 2016

14. Netlib; "HPL - A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers", http://www.netlib.org/benchmark/hpl/, hosted by The Innovative Computing Laboratory, University of Tennessee

15. Quinn; Parallel Programming in C with MPI and OpenMP, (2004) McGraw-Hill Companies, Inc. New York, New York, US

16. Saad; Interative Methods for Sparse Linear Systems (2nd ed); DOI: 10.1137/1.978089871800 (2003) SIAM, Philadelphia, PA, US

17. Snir, Otto, Huss-Lederman, Walker, Dongarra; MPI-The Complete Reference vol 1, The MPI Core (2nd ed), (1998) MIT Press, Cambridge, Mass., US

18. Supalov, Semin, Klemm, Danken; Optimizing HPC Applications with Intel(R) Cluster Tools, (2014) Springer's Open Access, DOI: 10.1007/978-1-4302-6497-2

19. Szekeres, L.; Payer, M.; Wei, L. T.; Sekar, R. (May 2014) "Eternal War in Memory", IEEE Security and Privacy, 1540-7993/14 © 2014 IEEE, pp 45-53, DOI: 10.1109/msp.2014.44

20. Texas Instruments, "SCAA082 High-Speed Layout guidelines" `www.ti.com/lit/an/scaa082/scaa082.pdf` (2006)

DOI: 10.14529/jsfi170205

Core Module Optimizing PDE Sparse Matrix Models with HPCG Example

Earle Jennings[1]

This paper introduces a fundamentally new computer architecture for supercomputers. The core module is application compatible with an existing superscalar microprocessor, with minimized energy use, and is optimized for local sparse matrix operations. Optimized sparse matrix manipulation is discussed by analyzing the High Performance Conjugate Gradient (HPCG) benchmark specification. This analysis shows how the DRAM memory wall is removed for this benchmark, and for sparse matrix models of partial differential equations (PDEs) for a wide cross section of applications. By giving the programmer improved control over the configuration of the supercomputer, the potential for communication problems is minimized. Application compatibility is achieved while removing the superscalar instruction interpreter and multi-thread controller from the existing microprocessor's hardware. These are transformed into compile-time utilities. The instruction cache is removed through an innovation in VLIW instruction processing. The data caches are unnecessary and are turned off in order to optimally implement sparse matrix models.
Keywords: *HPCG, superscalar, sparse matrix, partial differential equation, memory wall, HPC.*

Introduction

Today's high performance, superscalar microprocessor includes a superscalar instruction interpreter [12], instuction and data caches, as well as a multi-thread controller [19]. These elements consume more than 90% of the silicon and more than 90% of the energy, without processing any of the data. These are the energy monsters in today's high performance microprocessors. This paper introduces a core module known as the Simultaneous Multi-Processor (SiMulPro) core module, which removes all of these problems.

Application Compatibility

Application compatibility requires the existing compiler technology remain basically untouched. Removing the hardware overhead of the superscalar instruction interpreter, multi-thread controller and the instruction cache, requires that the SiMulPro core module be semantically compatible with the existing microprocessor of Figure 1. This means that each assembly language program needs to generate two applications, the first for the microprocessor, which is the first target of existing software tools. The second target is the SiMulPro core module, which does not implement the microprocessor's Instruction Set Architecture (ISA). This is discussed in section 1. Semantic compatibility is established for this assembly language program when both applications respond to the same input stream by generating essentially equal output streams.

Confirming application compatibility needs to be cost effectively managed by breaking this job into several steps. For High Performance Computers (HPC), particularly supercomputers, C, C++ and Fortran compilers are the most commonly used. Consider the C compiler. It has a compiler test set, which is used today to confirm generated assembly code targeting the microprocessor. The first step of verifying application compatibility uses this C compiler test set to verify semantic compatibility with the second target (SiMulPro core and core module) from its generated assembly language programs. The compiler test set is exercised through the C

[1]QSigma, Inc., Sunnyvale, USA

compiler to generate its family of assembly test programs. These generated assembly programs can verify semantic compatibility, and therefore application compatibility, between the existing microprocessor and the SiMulPro core module.

A second step uses the assembly code programs of one, or more, C function libraries, each with their verification set, to extend verification of semantic compatibility between these two targets. At this point, there are two ways to proceed. One approach continues to the C++ and Fortran compilers, their test sets, and so on. The other approach starts from the compiler output opcode range, and successively extends testing, using the verification and test suite of the microprocessor. The verification can extend beyond the compiler output opcode range, to include more, possibly all, of the ISA.

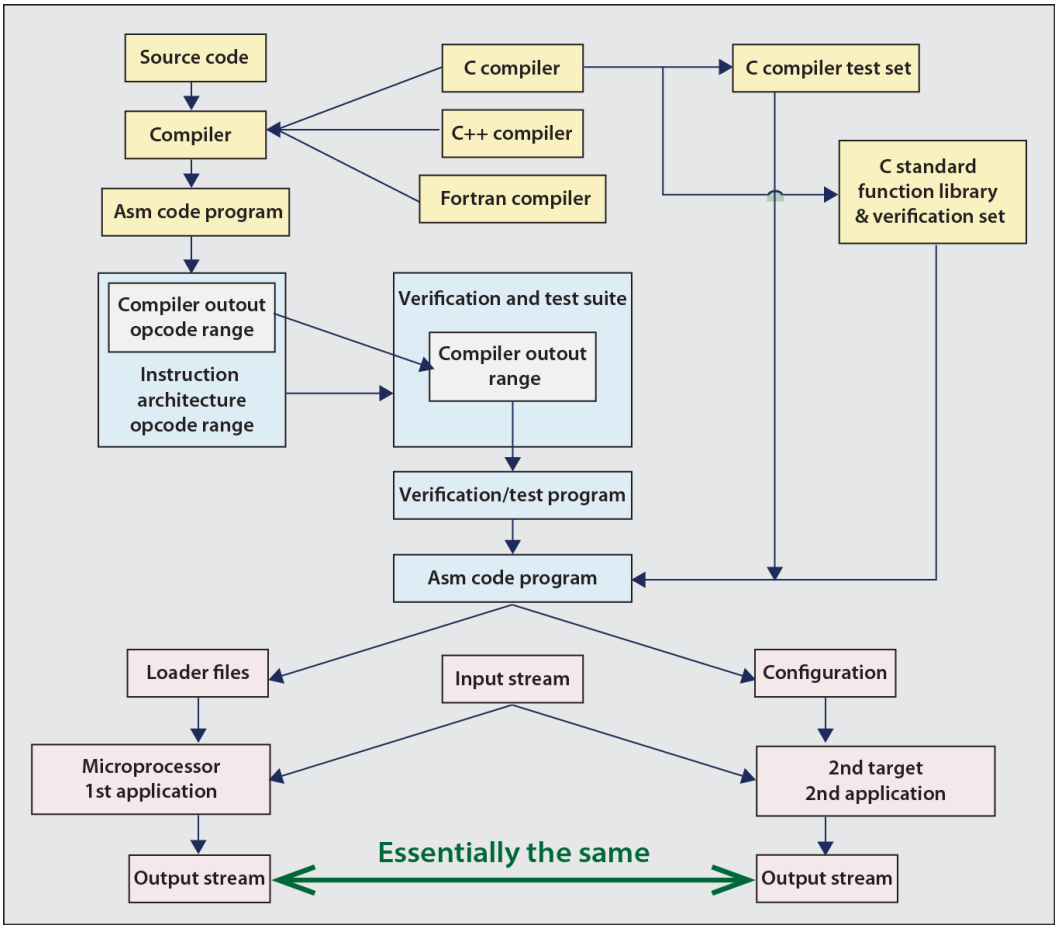


Figure 1. Semantic compatibility confirmed against both compiler technology and the verification and test set of the microprocessor implementing an Instruction Set Architecture (ISA)

Consider the verification and test set of the microprocessor of Figures 1 and 2. These verification and test sets are the primary technical collateral used to insure that going to silicon with the microprocessor's design is not a waste of money. This same verification and test set can be used in several ways to help remove the energy monsters from the microprocessor in an application compatible manner. What we have just described is the systematic and cost effective development of the semantic compatibility verification set for the second target. We will return to Figure 2 later, but first, the second target needs to be outlined.

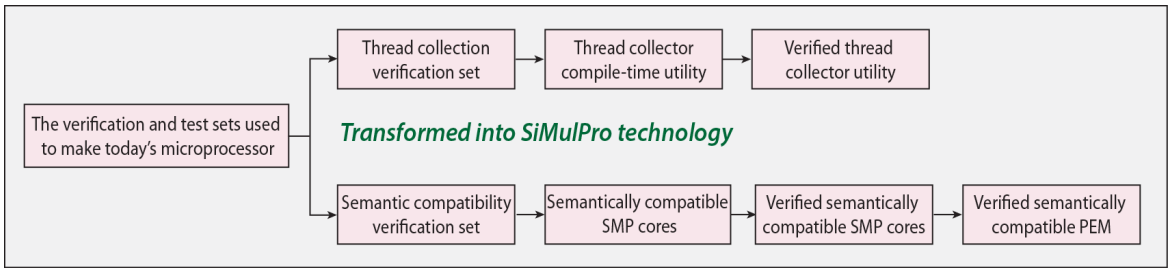


Figure 2. Components of the verification and test set and of the generated SiMulPro technology

1. Simultaneous Multi-Processor (SiMulPro) Cores And Core Module

This simple SiMulPro core supports two simultaneous processes. It implements a simultaneous process state calculator, which issues two process states for executing these processes.

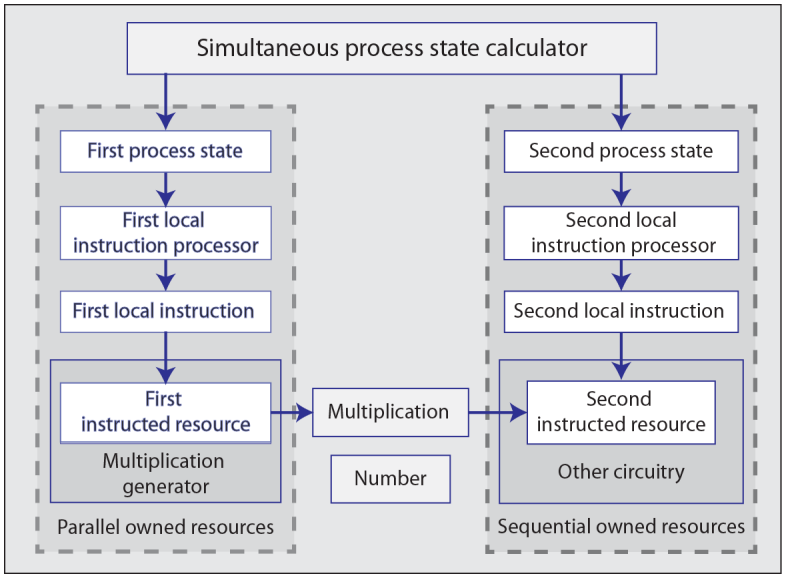


Figure 3. A simple SiMulPro core

Each of these simultaneous processes separately owns instructed resources. The separated ownership means that both processes cannot contend for the same resource, making them immune to stalling from contentions. Each instructed resource includes its own local instruction processor, which can access its own local instruction memory component. The state of each process stimulates each owned, local instruction processor. Each instruction processor accesses its own local instruction memory to generate the local instruction to direct that resource. Each of the processors includes its simultaneous process and its owned resources.

Figure 4 shows a use of this simplified SiMulPro core. The right hand side is characteristic of the Multiflow [6], and the EPIC architecture [16], which led to the I-64 [11] and the Itanium [17]. The Mill computer [7] bears some resemblance to the left hand side, but has only two instruction pointers.

The SiMulPro cores support factoring algorithms into their natural units of up to six, or more, simultaneous processes sufficient to handle the factoring of at least the algorithms as outlined in **Numerical Recipes** [14]. This large virtual VLIW space removes the need for

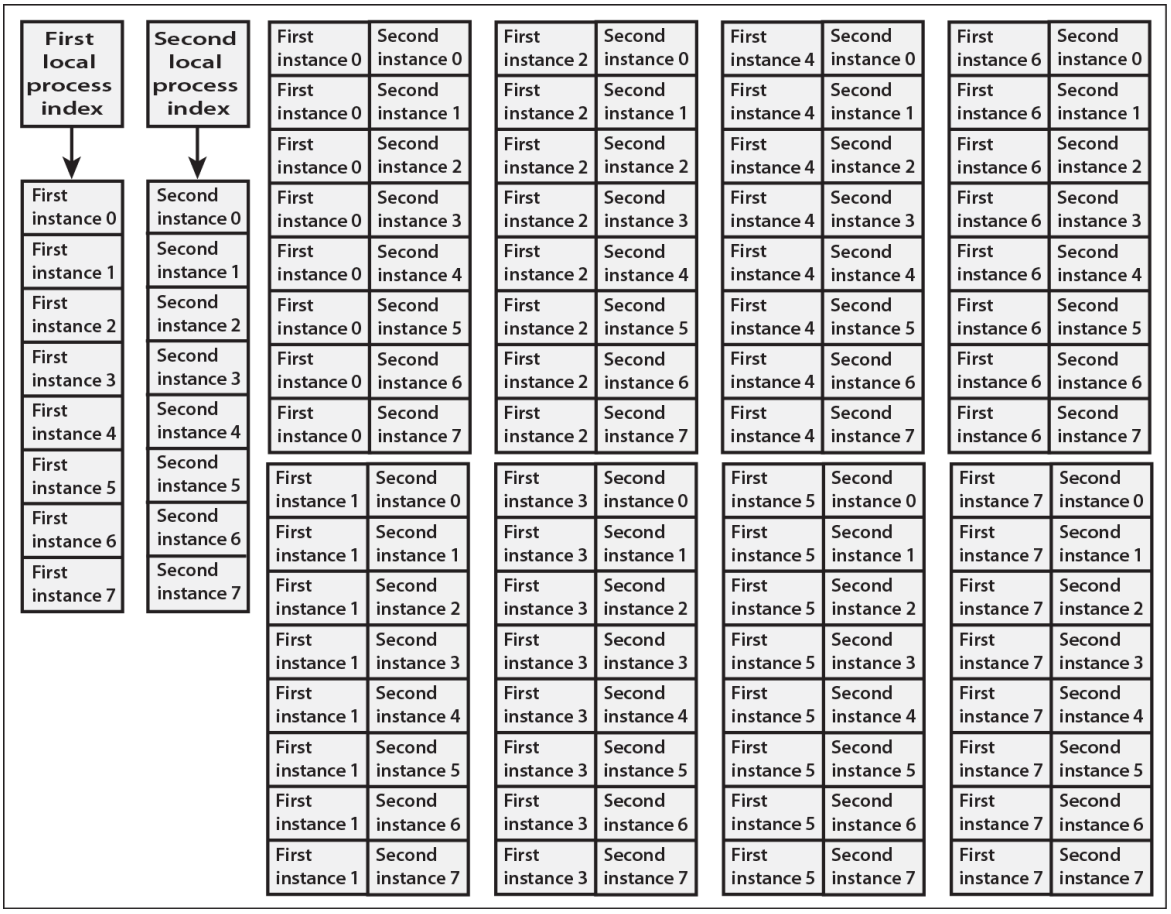


Figure 4. The first and second SiMulPro processes each have 8 process states, shown on the left. A single VLIW instruction pointer requires an equivalent VLIW memory of 64 VLIW words, shown on the right

instruction caching. Also, all predecessor VLIW approaches require unique compilers, which negates application compatibility.

Figure 5 shows the FP SiMulPro core including the process state calculator, which generates multiple process states (or indexes) and their corresponding loop outputs, on every clock cycle. This is shown in instruction pipe 0. These process states, etc., form an execution wave front, which successively traverses each instruction pipe. Assume each instructed resource includes 256 local instructions per task. If K is 4, this is a virtual VLIW instruction space of 4 Giga-VLIW instructions. If K is 6, this is 256 Tera-VLIW instructions.

Pipe 1 includes the Memory Access Processor (MAP), which generates all local addressing for many standard access patterns, including matrix and vectors, buffering, and FFTs. During local, linear algebra operations, as in Linpack, all non-floating point cores are turned off. The system capabilities of instruction pipes 2 and 4 are coupled:

- The FP Rams are organized so that a program can access them for complex numbers as 4 blocks, but are implemented as 8 blocks of 512 words. The read ports are in instruction pipe 2 and the write ports are in pipe 4. These can be programmed to limit access collisions for local, sparse matrix operations, to about 2% of the time. Each pair of read ports shares two queues, so these collisions do not significantly stall the multipliers in performing sparse matrix operations.

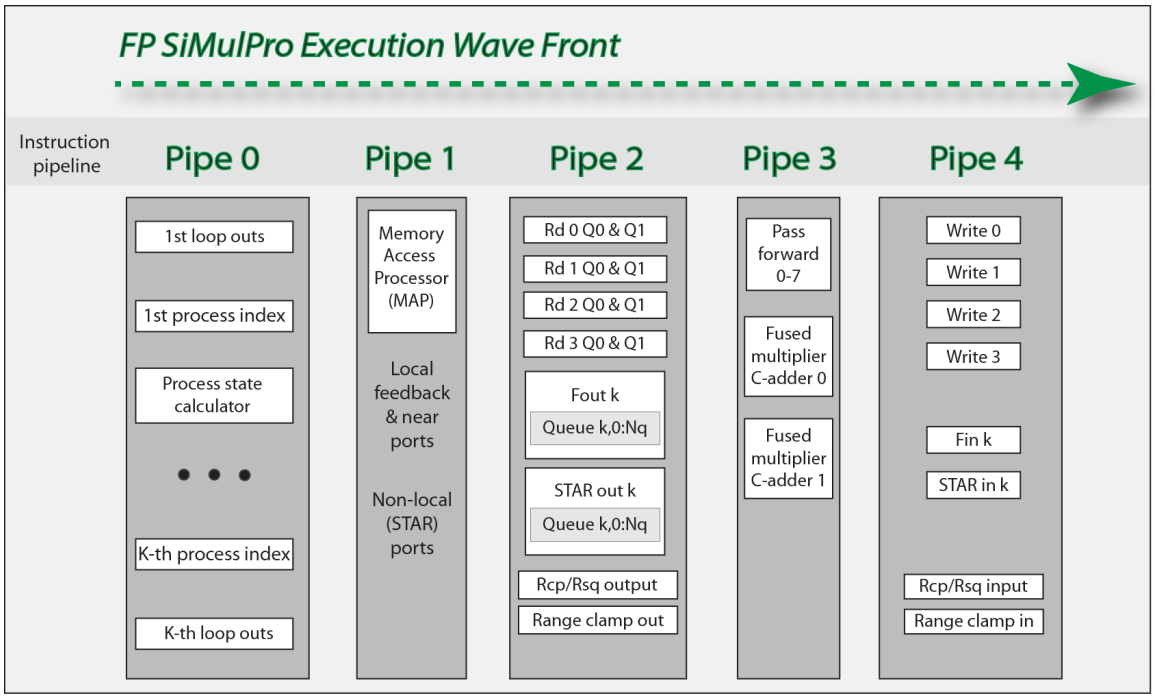


Figure 5. Example FP SiMulPro core supporting K simultaneous processors

- Data is fed back from instruction pipe 4 to output queues in instruction pipe 2. This feed mechanism also provides a router-less feed interface to neighboring modules. This is more energy efficient, because hardware routers are not needed.
- Farther communication uses the STAR (Simultaneous Transmit And Receive) input port of pipe 4 and output queues of pipe 2. The transfer request channel sends to a memory controller the parameters for each programmed memory access pattern. The state of the core modules can be saved with an acceptable overhead. The memory controllers can respond to anticipating memory requests, which preload data into the controller for low latency transfers when the data is actually needed. This combined with the VLIW instruction space makes caches unnecessary. Each channel can transmit and receive a STAR message on each local clock cycle, assumed to be 1 ns. The application layer of the STAR message includes a package with a payload of 128 + 5 bits and a context field of 32 bits. The payload can hold two double precision numbers, sufficient to efficiently download, or upload, the state of the cores in a Data Processing Chip (DPC). Also, the payload can hold a double precision number and an index list, which can represent a non-zero entry in a sparse matrix, or a vector component associated with one, or more, of the non-zero, sparse matrix entries. This package with, or without the context, is supported by the arithmetic and feed interfaces as well, so that sparse matrix processing, as well as matrix pivots, are consistently and efficiently supported.
- Reciprocals, reciprocal square roots and a range clamp circuit, for range limiting of transcendental functions, are each supported with input in pipe 4 and output in pipe 2.
- Integer to float and float to integer are also supported, though they are not shown.

Pipe 3 includes pass forward circuits and two instances of a fused multiplier C-adder. Semantic compatibility often requires a fused multiply-accumulate capability in Floating Point (FP) [5]. Fused multiplier Comparison (C)-Adders support not only FP, but also posit arithmetic [9] in at least 3 precisions, 64, dual 32, and quad 16 bit formats. With these capabilities,

algorithms can go from coarse (16 bit), to finer, and finest arithmetic (64 bit posit), with reduced communication overhead, and memory access for early iterations. Neural networks and deep learning can operate in 16 bit mode, providing 8 posit multiply-accumulates per execution wave front.

Within each execution wave front, a use vector, of the used resources is generated. Within the wave front, a use bit drives a power gate generating a gated power used by the resource, for each resource. In CMOS, the clock may be gated to control power. However, in technologies such as memristors and molecular gates, other specific mechanisms may provide this capability.

2. Core Modules And Programmable Execution Module (PEM)

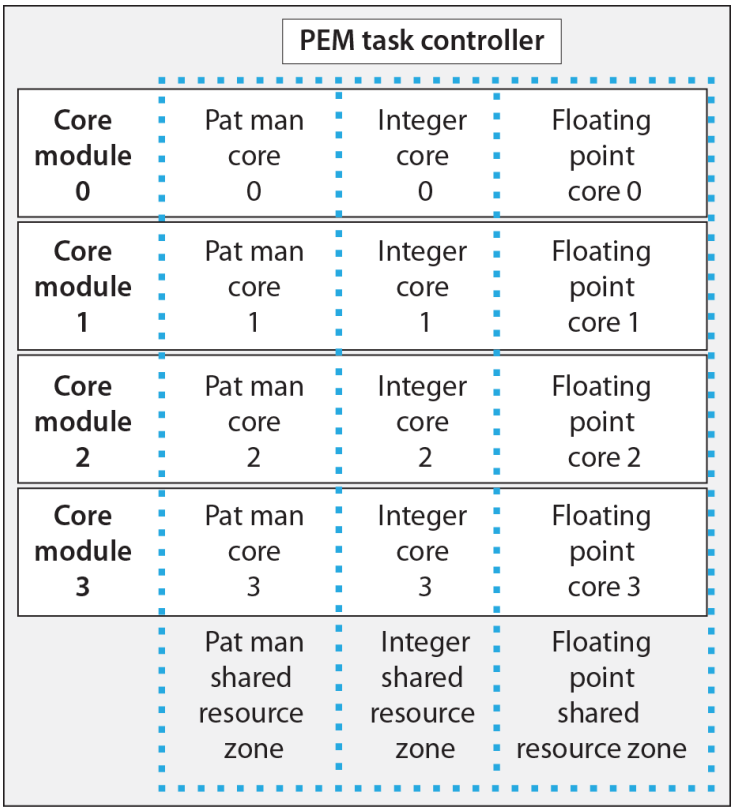


Figure 6. The PEM contains 4 core modules, each including 3 types of SiMulPro cores

The PEM is roughly a quad core superscalar microprocessor, without the overhead. Note that without this overhead Data Processor Chips (DPCs) can scale from about 60 core microprocessors to ten times that many core modules, with about the same power and silicon as today's chips. Instances of the same typed cores, such as the FP cores discussed above, can share their instructed resources. This increases the virtual VLIW space. Assume each FP core supports 6 simultaneous processors and 256 instructions per resource. The VLIW instruction space is then $2^{(4*6*8)} = 2^{192} = 2^2 2^{(19*10)} > 4 * 10^{57}$.

There are three types of cores in each PEM, FP, integer and Pattern Manager (Pat Man) cores. The integer core implements the arithmetic required for application compatibility, and also keeps track of the indexing associated with every word of the FP memory used for local sparse matrix oprations. However, the integer core does not generate main memory addressing. Access requests can be sent by the STAR input port to an external, anticipating memory controller for

DRAM. This saves at least 15% of the energy used in the core module. Pat Man organizes and directs the FP and integer activities needed for local sparse matrix operations.

Each core module can operate as a seperate unit. Core module 0 may perform a simulation of one object. Simultaneously, core module 1 may simulate a second model. Alternatively, each core module may be configured to model the same object, but at differing geometric locations.

3. Software Development Today And Tomorrow

The compiler is basically unchanged between today and tomorrow. The superscalar interpreter hardware, in particular its behavioral model, is transformed into the thread collector utility as shown in Figures 2 and 6. The thread collector is now a compile-time utility receiving assembly code, which it translates into microcode. The microcode is then scheduled, as close to the start of a thread as allowed by the preceding assembler instructions. The thread collector outputs these sequenced micro code instructions as one, or more, thread source code files.

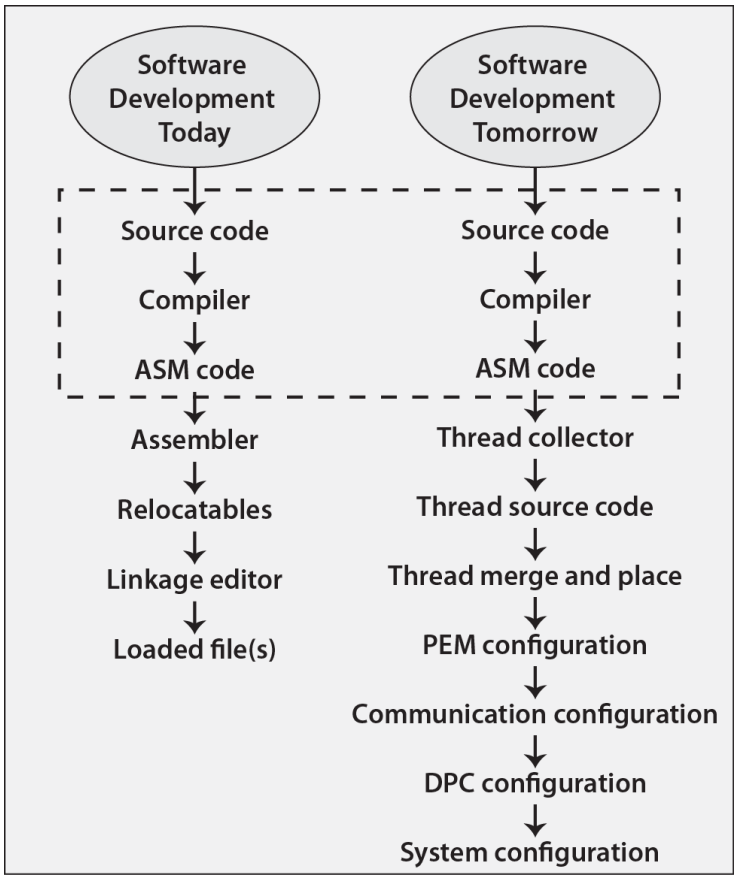


Figure 7. Software Development Summary

The multi-thread controller, becomes a software utility, which merges and places the threads into configurations of the PEM. The farther (STAR) network and the router-less neighbor communications are then configured for each DPC in the system. Configuring the system also involves configuring the anticipating memory controllers, interfaced to the DRAM, configuring communications within the DPC to across the system, and configuring communication with the data center, in which the supercomputer resides. The verification and test sets for today's superscalar instruction interpreter become the verification set for the thread collector. Exercising this verification set and the thread collector, generates the verified thread collector. The behav-

ioral models of the microprocessor's data processing resources are injected into SiMulPro core templates to create the core module. The semantic compatibility verification set is derived, as discussed above, from the compiler test sets, etc.. Exercising the core model with the semantic compatibility verification set generates the verified, semantically compatible SiMulPro cores and core module. Verifying the core module creates the semantically compatible PEM.

3.1. Threads

Today, a thread of execution, is the smallest sequence of program instructions which can be managed independently by a scheduler in an operating system. In this architecture, threads are operations of at least one core, in a PEM, expressed as one or more processes. Consider the following example shown in Figure 8 of a process defined in a thread source code file.

```
process DotAccum    // Declarations
    FeedbackInput
        ProductIn 1,        DotAccum 2;
    FeedbackOut
        ProductFedback1     ProductIn 1;
        DotFedback[1:5]     DotAccum[1:5];
    C-Adder                 DotAccum 0;
    STAR    Out Data        DotOut 0;
    //    Process List
    ProcessStates
        DPaccum6,    // highest priority, least probable process state
        DPaccum5, DPaccum4, DPaccum3, DPaccum2, DPaccum1,
        DPaccum0;    // least priority and most probable state
    endProcessStates;
Endprocess;
```

Figure 8. Short example of thread source code

Instructed resource	Dot product accumulate process	Filter one process	Filter two process	Calculate maximum process
In queues				Max-in queue
Feedback queues	Product fdbk 1, Dot accum 1 to n			Max-fdbk 2 to max-n
Memory read queue		Tap Read, Fir in	FFT-coef read, FFT-pass data	
Multiplier				
C-adder 0	Yes	Yes	Yes	Yes
C-adder 1				
Memory write ports		Fir-write accumulate		
Feedback in	Dot accumulate Feedback in	FIR accumulate	FFT accumulate	C Max Feedback in
Output portal	Yes	Yes	Yes	Yes

Figure 9. Resource ownership of four processes to be merged

After collecting the threads, the intermediate program representation is analyzed from the thread source code files, and thread merging begins. Consider the following example of processes shown in Figure 9, all to be placed in the same core after merging. Note that these four processes share ownership of C-adder0, as they all use this resource. Consider how actions are triggered by a simultaneous process. Each process owns all of its stimulators, which can be queues associated with one or more RAM read ports, feedback, feed local or feed farther (STAR) sources. The process can also be stimulated by the state of internal loop registers. Assume that the four processes of Figure 9 are to be merged and have the following process states:

Table 1. The states of the processes to be merged

Dot Accum	Filter 1	Filter 2	Max Calc	Merged
DPaccum6	Facc2	F2acc3	Max5	
DPaccum5	Facc1	F2acc2	Max4	
DPaccum4	Facc0	F2acc1	Max3	
DPaccum3		F2acc0	Max2	
DPaccum2			Max1	
DPaccum1			Max0	
DPaccum0				

Given that the highest priority state of each process (shown in the top row) is its least probable state, merging there processes proceeds by first listing the highest priority states of each process as shown in the Merged 1 column:

Table 2. Resulting top states of the merged process

Dot Accum	Filter 1	Filter 2	Max Calc	Merged 1	Merged 2
DPaccum6	Facc2	F2acc3	Max5	DPaccum6	DPaccum6
DPaccum5	Facc1	F2acc2	Max4	Facc2	Facc2
DPaccum4	Facc0	F2acc1	Max3	F2acc3	F2acc3
DPaccum3		F2acc0	Max2	Max5	Max5
DPaccum2			Max1		DPaccum5
DPaccum1			Max0		Facc1
DPaccum0					F2acc2

The merging of these processes continues in this fashion as shown in the Merged 2 column. At runtime, the instruction set operations are gone, replaced by their microcode sequences, scheduled as opportunistically as the assembly program segment permits, and then, possibly, merged with other simultaneous processes. This runtime situation is new to application developers. They need to know what processes have been initiated.

Table 3 shows a sketch of a runtime tool to aid programmers. The thread condition register can show a log of which processes have been initiated in a thread and what test conditions are being responded to by these processes. When the current thread has completed initiation,

Table 3. Example of a thread condition register

Thread Condition Register				
Collapse Count	Cond 0	Cond 1	Cond ...	Cond p

it releases its processes to respond to feedback, input and loop conditions. For instance, each thread may use a 2 bit arithmetic condition code, where '00' means == 0, '01' means >0, '10' means <0, and '11' means unavailable. Arithmetic error codes may be represented by another 2 bit code, where '00' means normal, '01' means infinite generated, '10' means Not A Number (NAN) generated, and '11' means unavailable.

```
subprogram X1(parm list) {
    code 1           // cond state length = 0  Cond collapse = 0
    if test1         //         0                          0
    { code 2)        //         1                          0
    else { code 3    //         2                          0
    } code 4         //         1                          1
}
```

Figure 10. Example showing condition length and collapse count of the condition code register

Let's look briefly at what is needed to aid the programmer in this post-branching architecture. We need a semantic content that consistently describes branching across Fortran, C and C++, at a minimum. Let's focus on Arithmetic IFs from Fortran, the switch conditional from C, and Range limiting for transcendental functions. Each of these branching mechnisms can be described by 2 bit encoding of trinary condition states, making it a feasible and consistent choice across all three languages. We also need some form of loop constructs that account for the looping of these languages. Fortunately, C provides us with two primitives into which the other implementations can be mapped:

```
do    {    body1    }  while    test1;
while      test2        do   {   body2    };
```

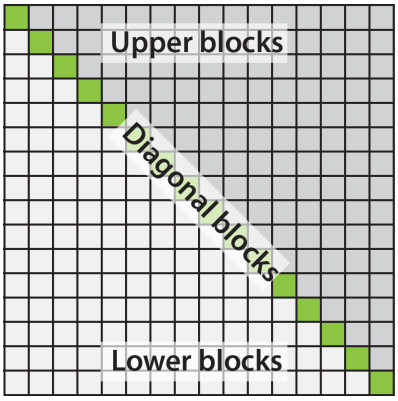


Figure 11. Thread Placement for Block LU Decomposition in one Data Processor Chip

Suppose a DPC is allocated to perform linear algebra operations such as Block LU decomposition, by placing three kinds of threads, lower block, upper block, and diagonal block processors [8], as shown in Figure 11. This is a very simplified presentation, placing only three

distinct threads. However, each core, in each of the PEM, can be configured differently. Complex parsers, data analytics, numerically intensive, and all other manner of threads, can be placed to create networks of state machines. These networks can be unique to each DPC. These state machine networks optimize the big computational, and the big data, processing of tomorrow.

4. Introduction To Sparse Matrix Mathematics And Models

A matrix is a 2-D rectangular array of numbers, either FP or posit numbers. A sparse matrix $A = [a_{i,j}]$ is a matrix in which most numbers are zeros. A column vector $x = \begin{bmatrix} x_0 & \dots & x_N \end{bmatrix}^T$ is called a stimulus and a vector $b = \begin{bmatrix} b_0 & \dots & b_N \end{bmatrix}^T$ is called a response when $Ax = b$, or $\sum a_{i,j}x_j = b_i$ for all i.

Matrix mathematics [8] is an innately useful tool in many applications [14]. In particular, sparse matrix math [15] has been harnessed to approximate solutions to partial differential equations [1], [18] specifically, 3-D models of physical systems, which includes fluids, stress and strain of solids, the dynamics of molecules, plasmas [2], as well as the weather.

Many differential operators, for example the Laplacian, support very stable and efficient finite difference approximations, such as the 27 stencil model [13] used in HPCG benchmark [10].

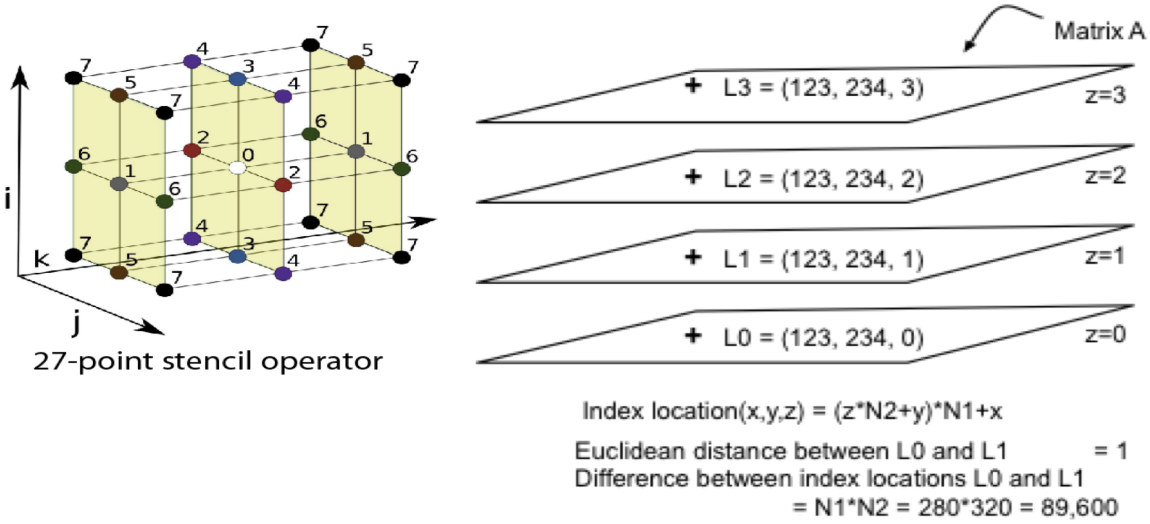


Figure 12. On left is the 27 point stencil from the HPCG specification, and on right, the index enumeration problem for finite difference schemes

The standard approaches to enumerating the finite difference-derived linear equations of the stencil involve traversing in one direction first. For example, along the i axis, then the j axis, followed by the k axis. This scatters the geometric locations across a wide section of the stimulus vector components. HPCG is a typically sized grid of 280 by 320 by 540 nodes. Consider the two neighboring points L0 and L1 on the right side of Figure 12. The Euclidean distance between L0 and L1 is 1, but distance between these index locations in the corresponding stimulus vector components, is 280*320 = 89,600 double precision numbers, or 716,800 bytes apart from each other. Note that two points separated by one in the j direction are 2,240 bytes apart. If two points are separated by two numbers in the j direction, this is enough to trigger a cache fault when the cache page size is 4096 bytes. Every time the sparse matrix row interacts with the stimulus vector at these two points, components of the stimulus vector are fetched at these greater distances.

These fetches of the stimulus vector are likely to trigger cache faults. To execute these sparse linear models efficiently, we need to situate the geometric neighborhoods of the model system in a single core. By concentrating the relevant geometric information in one core, data caching is unneeded. The main memory is accessed only once, eliminating the memory wall problem.

5. Local Implementation Of Sparse Matrix Solvers For A Geometric Neighborhood

The indexing rasterization discussed above for the 3-D model of HPCG, leads to an index location calculated as

$$IndexLocation\left(\begin{array}{cccc} x, & y, & z \end{array}\right) = (z * N2 + y) * N1 + x = IL$$

implying a localization function, which generates x, y, and z by performing

$$\begin{aligned} k &= floor\,(IL/N1) \\ x &= rem\,(IL/N1) \\ y &= rem\,(k/N2) \\ z &= floor\,(k/N2) \end{aligned}$$

Note that HPCG sets $N1 = 280$ and $N2 = 320$.

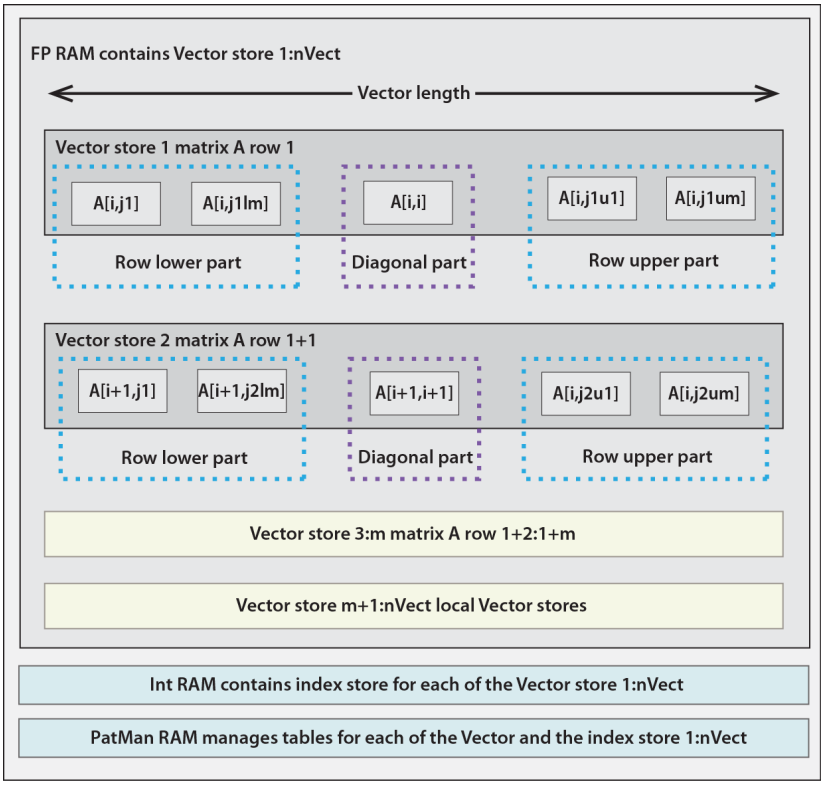


Figure 13. Allocation of FP RAM in Vector stores with the indexing mirrored by Int RAM

Figure 13 shows the FP RAM holding all the non-zero entries for several rows of the matrix A. Each of these rows includes a lower part, a diagonal part and an upper part. The relevant entries of each vector are stored locally in vector stores in the FP RAM. Each of the stimulus vector entries corresponds with (at least) one of the non-zero entries of the A matrix stored

locally. The response vector components correspond to one of the row indexes of the matrix rows being stored locally. The vector length is chosen so that the stencil can be stored in just one vector store, and the boundary stencils fit naturally within this vector store, without requiring garbage collection. The integer RAM contains the indexes of the corresponding FP RAM locations. The Pat Man core keeps track of the management tables for the FP and Int Cores. Note that in some implementations the vector stores for the stimulus and response vectors may be organized differently.

6. Introduction To The Pat Man Core

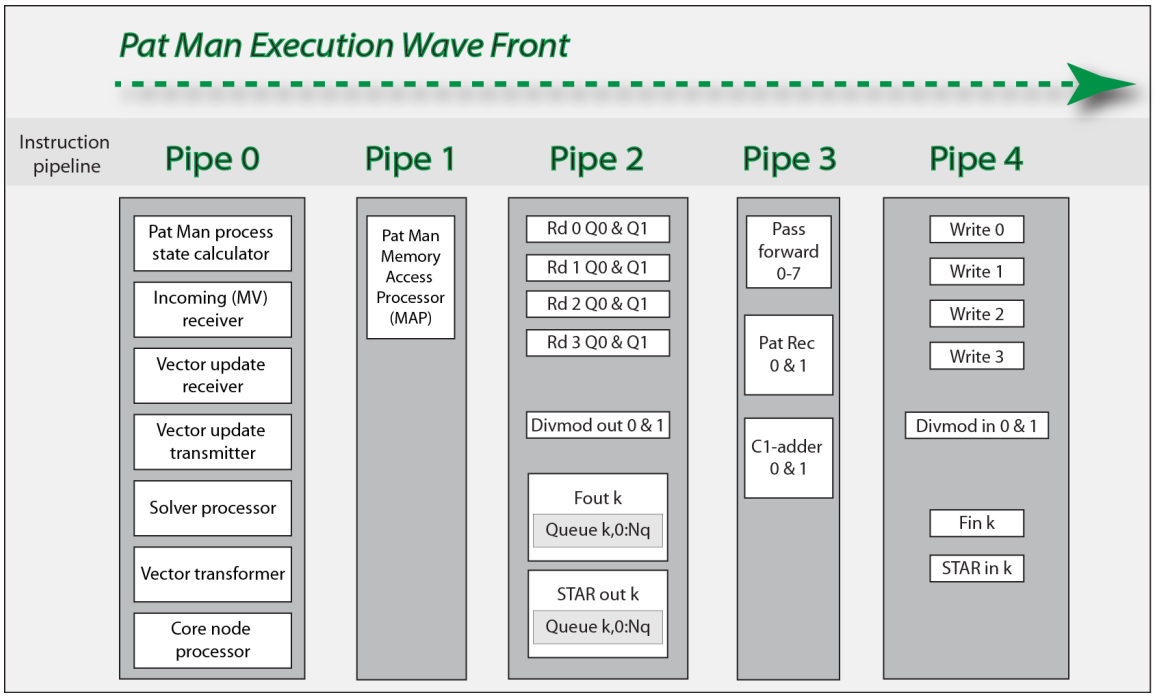


Figure 14. Block diagram of the Pat Man core

The Pat Man core shown in Figure 14 is a SiMulPro core, with the following components.

1. The C1-adders of pipe 3 match the index list of a received package to determine which global object is being referenced, whether the package should be stored in this core module, and whether the global object uses finite difference, or finite element, indexing. Often, finite element indexing is organized to reveal geometric proximity.
2. Divmod in and Divmod out of pipes 4 and 2, respectively implement various schemes for 2-D, 3-D (shown above) index location conversion to geometric address. Higher dimensional indexing may also be supported.
3. Pattern Recognizers (PatRecs) use either the index list (for finite element indexing), or the derived geometric addresses, to determine which vector store holds the package numeric data.

The sparse matrix A is downloaded to the relevant DPCs, where these downloaded entries are processed for storage in their specific core modules. Once completed, the vectors are initialized, with the core modules storing the relevant components for their model locally. At this point in each core module, the Pat Man core's determine which row, or rows, can be processed to alter the vector components. The altered components are then distributed as needed by one,

or more, of the communication mechanisms. The received vector updates are used by the Pat Man cores to update the readiness of the local sparse rows for processing. When a row is ready for processing, the PatMan sends a message to the FP and Int cores, which is queued until the cores are ready to begin those operations.

7. Basic Systems Analysis Of Sparse Matrix Performance

While HPCG is not a real program solving one or more partial differential equations, several things can be done to evaluate what will happen to other application models of similar size. Each of the FP RAM includes 4K words, which supports 151 vector stores. If 25% of these stores are allocated to the local components of the vectors, each core module can execute about 110 rows of the sparse matrix. The 48,384,000 rows of HPCG, and comparable models, fit in 440K core modules, or about 764 Data Processor Chips (DPCs), which each contain 144 PEM, or 576 core modules. Assuming 16 DPC chip stacks per optical PCB and 12 PCB's per optical motherboard, this is essentially a single rack containing 4 optical motherboards. This rack, once loaded from the DRAM, never needs to access the DRAM for the matrix values, or vector values, until the run is over, or the DRAM needs updating due to adaptive grid generation, or refinements of the finite element cells (etc.).

The remaining overhead is primarily the communication overhead for vector updates. Consider a worst case situation for HPC. Suppose a tokamak reactor containing plasma is simulated as the interior of a torus as shown in Figure 15. This is a very simplified discussion, which will not require detailed knowledge of plasma physics nor the specifics of a particular solution algorithm for such equation systems. Simply assume this model saturates a supercomputer covering a computer floor of roughly 40 meters on a side. Further assume the torus is modeled as a sheet of core modules, which has its top and bottom edges (ab and cd) joined, then its left and right edges joined, which is a classic topological construction of the torus. This naive construction has just condemned the simulation to have a huge energy budget, as well as very poor communication latency, because the body effect latency now includes the worst case propagation from the bottom left to the top right cabinets.

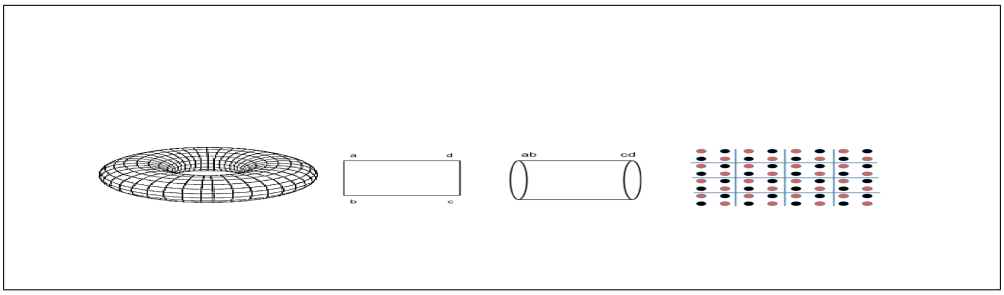


Figure 15. Making a torus by identifying sides, and then imposing a 2-D red-black ordering

To bring horizontal traversal under control, rather than enumerate the local geometric neighbors successively, employ a red-black ordering scheme so that the first half of the right to left cells run from left to right horizontally as red cells, with the second half of these cells as black cells running from right to left. Now the farthest left and the farthest right neighborhoods can meet in two neighboring PEM. To remove the vertical traversal problem, further apply the red black scheme so that the first half of the local neighborhoods traverse from top to botton as red cells and the second half traverse from the bottom going to the top as black cells. Now, the top

cells meet the bottom cells within two neighboring PEM, and the worst case traversal path is just from one cabinet to an adjacent cabinet.

8. Summary

A fundamentally new computer architecture has been introduced. This architecture is application compatible with an existing superscalar microprocessor, which can be verified in a systematic, incremental approach lending itself to effective project management. The superscalar instruction interpreter and multi-thread controller are removed from the microprocessor and transformed into compile-time utilities. Today's compilers for the microprocessor are preserved, essentially unaltered. The instruction cache is unneeded because of the huge, virtual VLIW space. Sparse matrix modeling of partial differential equations and HPCG, are optimized by removing the DRAM memory wall. Communication latency and throughput can be controlled by thread placement, as shown by the tokamak and block LU decomposition examples. New software tools, while maintaining today's compilers, enable programs to be embodied by these supercomputers as algorithm state machine networks.

Energy use is minimized. The superscalar instruction interpreter, the caches, and the multi-thread controller are removed from hardware, giving at least a 10X reduction. Each SiMulPro core, gates off power to each unused resource on each clock. Local feed communications with neighboring PEM do not use routers. Sending access requests to external, anticipating memory controllers for DRAM, saves at least 15% of the energy used for memory address calculation. Sparse matrix solvers, in systems as small as a rack, are only loaded once from the DRAM.

I wish to thank Heather Murphree, John Gustafson, and George Pearson of MacKickan Software for their invaluable contributions. I wish to thank the reviewer for pointing to an inconsistency in the discussion of sparse matrices. Thanks to the HPCG team for their thinking and their excellent bibliography. Also, thanks to the applied mathematics community for giving us all such powerful tools for partial differential equations.

References

1. Briggs, Henson, McCormick; A Multigrid Tutorial (2nd ed), DOI: 10.1137/1.9780898719505 (2000) Society of Industrial and Applied Mathematics (SIAM), Pihladelp[hia, PA, US

2. DOE; Scientific Grand Challenges in Fusion Energy Sciences and the Role of Computing at the Extreme Scale, `https://science.energy.gov/~/media/ascr/pdf/program-documents/docs/Fusion_report.pdf` (2009) DOE, Office of Fusion Energy Sciences, Office of Advanced Scientific Computer Research March 18-20, 2009, US

3. DOE-ASCAC Subcommittee Report; Top Ten Exascale Research Challenges, `science.energy.gov/~/media/ascr/ascac/pdf/meetings/20140210/Top10reportFEB14.pdf` (2014), US

4. Dongarra; Report on the Sunway TaihuLight System; (2016), University of Tennessee, Oak Ridge National Laboratory, Dept Electrical Engineering and Computer Science, Tech Report, UT-EECS-16-742, US

5. Ercegovac, Lang; Digital Arithmetic, https://www.elsevier.com/books/digital-arithmetic/ercegovac/978-1-55860-798-9 Hardcover ISBN: 9781558607989 , (2003) Elsevier Sciences, San Franciso, CA, US

6. Fisher; Very Long Instruction Word Architectures and the ELI-512, https://doi.org/10.1145/800046.801649, (1983) ACM, US

7. Goddard; Ivan Goddard and Mill at the 2015 European LLVM Conference, April 14, 2015; https://millcomputing.com/event/1725/

8. Golub, van Loan; Matrix Computations (4th ed); https://jhupbooks.press.jhu.edu/content/matrix-computations-0 ISBN: 9781421407944, (2013) Johns Hopkins University Press, Baltimore, Maryland, US

9. Gustafson; Beating Floating Point at its Own Game: Posit Arithmetic; http://supercomputingfrontiers.com/2017/wp-content/uploads/2017/03/2_1100_John-Gustafson.pdf (2017)

10. Heroux, Dongara, Luszcek; HPCG Technical Specification, SAND 203-8752, www.osti.gov/scitech/biblio/1113870f (2013), Sandia National Labs, US

11. Huck, Morris, Ross, Kneis, Mulder, Zahir; Introducing the I-64 Architecture, http://ieeexplore.ieee.org/document/877947/ (2000), IEEE Micro, Sept, 2000, pp 24-35, US

12. Johnson; Superscalar Microprocessor Design, https://books.google.com/books/about/Superscalar_microprocessor_design.html?id=9o1TAAAAMAAJ (1991), Prentis Hall, Englewood, NJ, US

13. O'Reilly; A Family of Large-Stencil Discrete Laplacian Approximations in Three Dimensions, ftp://grey.colorado.edu/pub/oreilly/misc/disc_lapl.3.pdf (2006) University of Colorado Boulder, CO, US

14. Press, Flannery, Teukolsky, Vetterling; Numerical Recipes in C: The Art of Scientific Programming 2nd ed., http://apps.nrbook.com/c/index.html (1992) Cambridge University Press, Cambridge, England

15. Saad; Interative Methods for Sparse Linear Systems (2nd ed); DOI: 10.1137/1.97808987180 (2003) SIAM, Philadelphia, PA, US

16. Schlansker, Rau; EPIC: An Architecture for Instruction Level Parallel Processors, (February 2000) www.hpl.hp.com/techreports/1999/HPL-1999-111.pdf HP Laboratories Palo Alto, HPL-1999-111

17. Sharangpani, Arora; Itanium Processor Microarchitecture, https://www.researchgate.net/publication/3215154_Itanium_processor_microarchitecture (2000), IEEE Micro, Sept-Oct 2000, pp 24-43, US

18. Trottenberg, Osterlee, Shueller, Stuben, Oswald, Brandt; Multigrid; https://books.google.com/books/about/Multigrid.html?id=9ysyNPZoR24C (2001) Academic Press, Harcourt Science and Technology Company, San Diego, CA, US

19. Ungerer, Robic, Silc; A Survey of Processors with Explicit Multi-Threading, http://www.academia.edu/26319932/A_survey_of_processors_with_explicit_multithreading (2003), ACM Computing Surveys, ACM, vol. 35, No 1, March 2003, pp 29-63, US

DOI: 10.14529/jsfi170206

Beating Floating Point at its Own Game: Posit Arithmetic

John L. Gustafson[1], *Isaac Yonemoto*[2]

A new data type called a *posit* is designed as a direct drop-in replacement for IEEE Standard 754 floating-point numbers (floats). Unlike earlier forms of universal number (unum) arithmetic, posits do not require interval arithmetic or variable size operands; like floats, they round if an answer is inexact. However, they provide compelling advantages over floats, including larger dynamic range, higher accuracy, better closure, bitwise identical results across systems, simpler hardware, and simpler exception handling. Posits never overflow to infinity or underflow to zero, and "Not-a-Number" (NaN) indicates an action instead of a bit pattern. A posit processing unit takes less circuitry than an IEEE float FPU. With lower power use and smaller silicon footprint, the posit operations per second (POPS) supported by a chip can be significantly higher than the FLOPS using similar hardware resources. GPU accelerators and Deep Learning processors, in particular, can do more per watt and per dollar with posits, yet deliver superior answer quality.

A comprehensive series of benchmarks compares floats and posits for decimals of accuracy produced for a set precision. Low precision posits provide a better solution than "approximate computing" methods that try to tolerate decreased answer quality. High precision posits provide more correct decimals than floats of the same size; in some cases, *a 32-bit posit may safely replace a 64-bit float*. In other words, posits beat floats at their own game.

Keywords: computer arithmetic, energy-efficient computing, floating point, posits, LINPACK, linear algebra, neural networks, unum computing, valid arithmetic.

1. Background: Type I and Type II Unums

The ***unum*** (***u***niversal ***num***ber) arithmetic framework has several forms. The original "Type I" unum is a superset of IEEE 754 Standard floating-point format [2, 7]; it uses a "ubit" at the end of the fraction to indicate whether a real number is an exact float or lies in the open interval between adjacent floats. While the sign, exponent, and fraction bit fields take their definition from IEEE 754, the exponent and fraction field lengths vary automatically, from a single bit up to some maximum set by the user. Type I unums provide a compact way to express *interval arithmetic*, but their variable length demands extra management. They can duplicate IEEE float behavior, via an explicit rounding function.

The "Type II" unum [4] abandons compatibility with IEEE floats, permitting a clean, mathematical design based on the *projective reals*. The key observation is that signed (two's complement) integers map elegantly to the projective reals, with the same wraparound of positive numbers to negative numbers, and the same ordering. To quote William Kahan [5]:

> "They typically save storage space because what you're manipulating are not the numbers, but *pointers* to the values. And so, it's possible to run this arithmetic very, very fast."

The structure for 5-bit Type II unums is shown in fig. 1. With n bits per unum, the "u-lattice" populates the upper right quadrant of the circle with an ordered set of $2^{n-3} - 1$ real numbers x_i (not necessarily rational). The upper left quadrant has the negatives of the x_i, a reflection about the vertical axis. The lower half of the circle holds *reciprocals* of numbers in the top half, a reflection about the horizontal axis, making × and ÷ operations as symmetrical as + and −. As with Type I, Type II unums ending in `1` (the ubit) represent the open interval between adjacent exact points, the unums for which end in `0`.

[1]A*STAR Computational Resources Centre and National University of Singapore (joint appointment), Singapore
[2]Interplanetary Robot and Electric Brain Company, Saratoga, California, USA

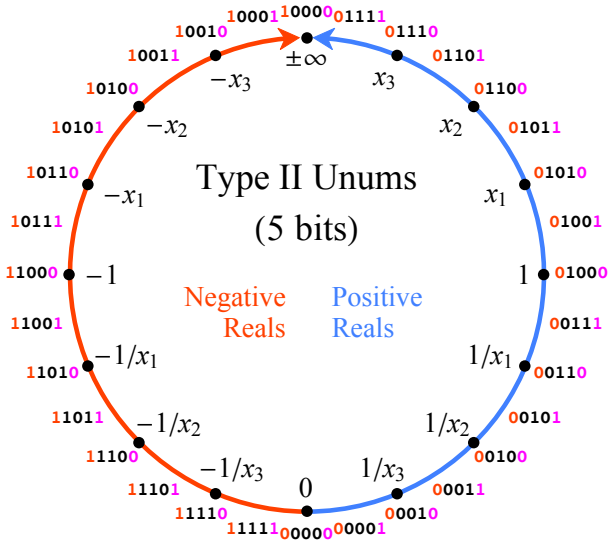


Figure 1. Projective real number line mapped to 4-bit two's complement integers

Type II unums have many ideal mathematical properties, but rely on **table look-up** for most operations. If they have n bits of precision, there are (in the worst case) 2^{2n} table entries for 2-argument functions, though symmetries and other tricks usually reduce that to a more manageable size. Table size limits the scalability of this ultra-fast format to about 20 bits or less, for current memory technology. Type II unums are also are much less amenable to *fused* operations. These drawbacks motivated a search for a format that would keep many of the merits of Type II unums, but be "hardware friendly," that is, computable using existing float-like logic.

2. Posits and Valids

We contrast two esthetics for calculation involving real numbers:

- Non-rigorous, but cheap, fast and "good enough" for an established set of applications
- Rigorous and mathematical, even at the cost of greater execution time and storage

The first esthetic has long been addressed by float arithmetic, where rounding error is tolerated, and the second esthetic has been addressed by interval arithmetic. Type I and Type II unums can do either, which is one reason they are "universal numbers." However, if we are always going to use the "guess" function to round after every operation, we are better off using the last bit as another significant fraction bit and not as the ubit. A unum of this type is what we call a *posit*. To quote the New Oxford American Dictionary (Third Edition):

> **posit** (noun): a statement that is made on the assumption that it will prove to be true.

A hardware-friendly version of Type II unums relaxes one of the rules: Reciprocals only follow the perfect reflection rule for $0, \pm\infty$, and integer powers of 2. This frees us to populate the u-lattice in a way that keeps the finite numbers *float-like*, in that they are all of the form $m \cdot 2k$ where k and m are integers. There are no open intervals.

A *valid* is a pair of equal-size posits, each ending in a ubit. They are intended for use where applications need the rigor of interval-type bounds, such as when debugging a numerical algorithm. Valids are more powerful than traditional interval arithmetic and less prone to rapidly expanding, overly pessimistic bounds [2, 4]. They are not the focus of this paper, however.

2.1. The Posit Format

Here is the structure of an n-bit posit representation with es exponent bits (fig. 2).

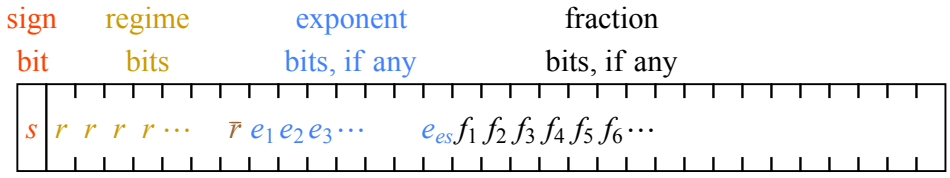


Figure 2. Generic posit format for finite, nonzero values

The sign bit is what we are used to: **0** for positive numbers, **1** for negative numbers. If negative, take the 2's complement before decoding the regime, exponent, and fraction.

To understand the *regime* bits, consider the binary strings shown in Table 1, with numerical meaning k determined by the *run length* of the bits. (An "x" in a bit string means, "don't care").

Table 1. Run-length meaning k of the regime bits

Binary	**0000**	**0001**	**001x**	**01xx**	**10xx**	**110x**	**1110**	**1111**
Numerical meaning, *k*	−4	−3	−2	−1	0	1	2	3

We call these leading bits the *regime* of the number. Binary strings begin with some number of all **0** or all **1** bits in a row, terminated either when the next bit is opposite, or the end of the string is reached. Regime bits are color-coded in **amber** for the identical bits r, and **brown** for the opposite bit $\bar{r}$ that terminates the run, if any. Let m be the number of identical bits in the run; if the bits are **0**, then $k = -m$; if they are **1**, then $k = m - 1$. Most processors can "find first **1**" or "find first **0**" in hardware, so decoding logic for regime bits is readily available. The regime indicates a scale factor of $useed^k$, where $useed = 2^{2^{es}}$. Table 2 shows example $useed$ values.

Table 2. Table 1. The *useed* as a function of *es*

es	0	1	2	3	4
useed	2	$2^2 = 4$	$4^2 = 16$	$16^2 = 256$	$256^2 = 65536$

The next bits (color-coded **blue**) are the exponent e, regarded as an unsigned integer. There is no bias as there is for floats; they represent scaling by 2^e. There can be up to es exponent bits, depending on how many bits remain to the right of the regime. This is a compact way of expressing *tapered accuracy*; numbers near 1 in magnitude have more accuracy than extremely large or extremely small numbers, which are much less common in calculations.

If there are any bits remaining after the regime and the exponent bits, they represent the fraction, f, just like the fraction **1**.f in a float, with a hidden bit that is always **1**. There are no subnormal numbers with a hidden bit of **0** as there are with floats.

The system just described is a natural consequence of populating the u-lattice. Start from a simple 3-bit posit; for clarity, fig. 3 shows only the right half of the projective reals. So far, fig. 3 follows Type II rules. There are only two posit *exception values*: 0 (all **0** bits) and ±∞ (**1** followed by all **0** bits), and their bit string meanings do not follow positional notation. For the other posits in fig. 3, the bits are color-coded as described above. Note that positive values in fig. 3 are exactly $useed$ to the power of the k value represented by the regime.

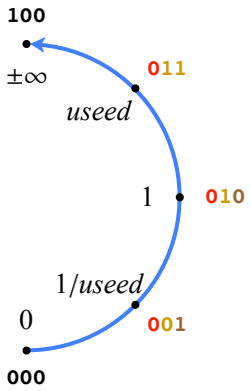


Figure 3. Positive values for a 3-bit posit

Posit precision increases by appending bits, and values remain where they are on the circle when a **0** bit is appended. Appending a **1** bit creates a new value between two posits on the circle. What value should we assign to each in-between value? Let *maxpos* be the largest positive value and *minpos* be the smallest positive value on the ring defined with a bit string. In fig. 3, *maxpos* is *useed* and *minpos* is $1/useed$. The interpolation rules are as follows:

- Between the *maxpos* and $\pm\infty$, the new value is $maxpos \times useed$; and between 0 and *minpos*, the new value is $minpos/useed$ (new **regime** bit).
- Between existing values $x = 2^m$ and $y = 2^n$ where m and n differ by more than 1, the new value is their *geometric* mean, $\sqrt{x \cdot y} = 2^{(m+n)/2}$ (new **exponent** bit).
- Otherwise, the new value is midway between the existing x and y values next to it, that is, it represents the *arithmetic* mean $(x + y)/2$ (new **fraction** bit).

As an example, fig. 4 shows a build up from a 3-bit to a 5-bit posit with $es = 2$, so $useed = 16$:

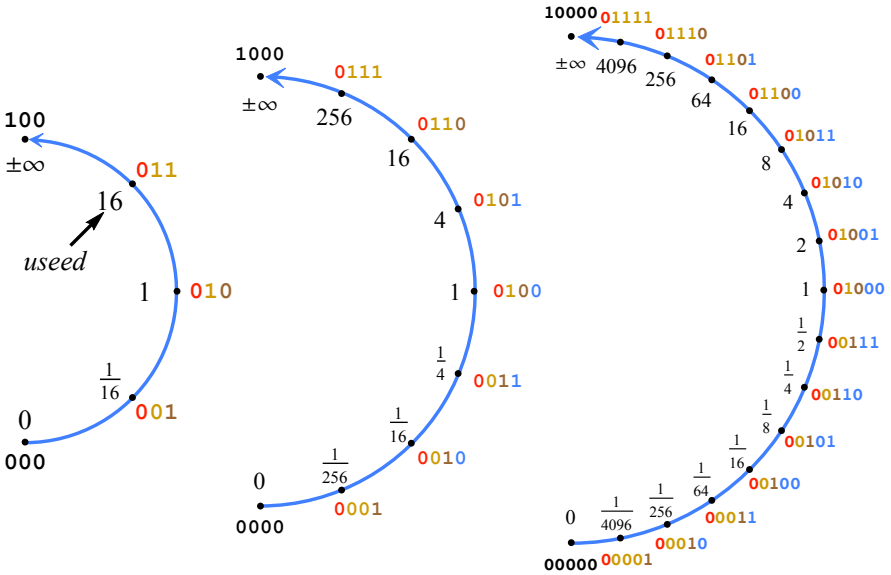


Figure 4. Posit construction with two exponent bits, $es = 2$, $useed = 2^{2^{es}} = 16$

If one more bit were appended in fig. 4 to make 6-bit posits, posits representing the range of values between 1/16 and 16 will append **fraction** bits, not exponent bits.

Suppose we view the bit string for a posit p as a signed integer, ranging from -2^{n-1} to $2^{n-1}-1$. Let k be the integer represented by the regime bits, e be the unsigned integer represented by

the exponent bits, if any. If the set of fraction bits is $\{f_1 f_2 \ldots f_{fs}\}$, possibly the empty set, let f be the value represented by $1.f_1 f_2 \ldots f_{fs}$. Then p represents

$$x = \begin{cases} 0, & p = 0, \\ \pm\infty, & p = -2^{n-1}, \\ sign(p) \times useed^k \times 2^e \times f, & \text{all other } p. \end{cases}$$

The regime and *es* bits serve the function of the exponent bits in a standard float; together, they set the power-of-2 scaling of the fraction where each *useed* increment is a batch shift of 2^{es} bits. The *maxpos* is $useed^{n-2}$ and the *minpos* is $useed^{2-n}$. An example decoding of a posit is shown in fig. 5 (with a "nonstandard" value for *es* here, for clarity).

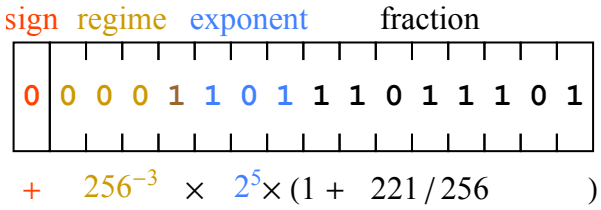


Figure 5. Example of a posit bit string and its mathematical meaning

The sign bit **0** means the value is positive. The regime bits **0001** have a run of three **0**s, which means k is -3; hence, the scale factor contributed by the regime is 256^{-3}. The exponent bits, **101**, represent 5 as an unsigned binary integer, and contribute another scale factor of 2^5. Lastly, the fraction bits **11011101** represent 221 as an unsigned binary integer, so the fraction is $1 + 221/256$. The expression shown underneath the bit fields in fig. 5 works out to $477/134217728 \approx 3.55393 \times 10^{-6}$.

2.2. 8-bit Posits and Neural Network Training

While IEEE floats do not define a "quarter-precision" 8-bit float, an 8-bit posit with $es = 0$ has proved to be surprisingly useful for some purposes; they are sufficiently powerful to *train neural networks* [3, 8]. Currently, half-precision (16-bit) IEEE floats are often used for this purpose, but 8-bit posits have the potential to be 2–4× faster. An important function for neural network training is a *sigmoid function*, a function $f(x)$ that is asymptotically 0 as $x \to -\infty$ and asymptotically 1 as $x \to \infty$. A common sigmoid function is $1/(1 + e^{-x})$ which is expensive to compute, easily requiring over a hundred clock cycles because of the math library call to evaluate $\exp(x)$, and because of the divide. With posits, you can simply flip the first bit of the posit representing x, shift it two bits to the right (shifting in **0** bits on the left), and the resulting posit function in fig. 6 (shown in **magenta**) closely resembles $1/(1 + e^{-x})$ (shown in **green**); it even has the correct slope where it intersects the y-axis.

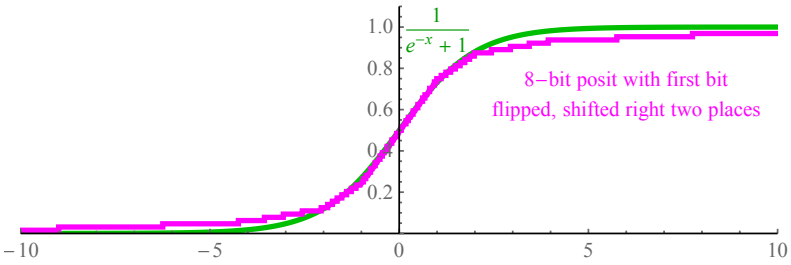


Figure 6. Fast sigmoid function using posit representation

2.3. Using the Useed to Match or Exceed the Dynamic Range of Floats

We define the *dynamic range* of a number system as the number of decades from the smallest to largest positive finite values, *minpos* to *maxpos*. That is, the dynamic range is defined as $\log_{10}(maxpos) - \log_{10}(minpos) = \log_{10}(maxpos/minpos)$. For an 8-bit posit system with $es = 0$, *minpos* is 1/64 and *maxpos* is 64, so the dynamic range is about 3.6 decades. Posits defined with $es = 0$ are elegant and simple, but their 16-bit and larger versions have less dynamic range than an IEEE float of the same size. For example, a 32-bit IEEE float has a dynamic range of about 83 decades, but a 32-bit posit with $es = 0$ will have only about 18 decades of dynamic range.

Here is a table of *es* values that allow posits to surpass the dynamic range of floats for 16-bit and 32-bit size, and closely match it for 64-bit, 128-bit, and 256-bit sizes.

Table 3. Float and posit dynamic ranges for the same number of bits

Size, Bits	IEEE Float Exp. Size	Approx. IEEE Float Dynamic Range	Posit *es* Value	Approx. Posit Dynamic Range
16	5	6×10^{-8} to 7×10^{4}	1	4×10^{-9} to 3×10^{8}
32	8	1×10^{-45} to 3×10^{38}	3	6×10^{-73} to 2×10^{72}
64	11	5×10^{-324} to 2×10^{308}	4	2×10^{-299} to 4×10^{298}
128	15	6×10^{-4966} to 1×10^{4932}	7	1×10^{-4855} to 1×10^{4855}
256	19	2×10^{-78984} to 2×10^{78913}	10	2×10^{-78297} to 5×10^{78296}

One reason for choosing $es = 3$ for 32-bit posits is to make it easier to substitute them not just for 32-bit floats, but for 64-bit floats as well. Similarly, the 17-decade dynamic range of 16-bit posits opens them up to applications currently are tackled only with 32-bit floats. We will show that posits can exceed both the dynamic range *and* accuracy of floats with the same bit width.

2.4. Qualitative Comparison of Float and Posit Formats

There are no "NaN" (not-a-number) bit representations with posits; instead, the calculation is interrupted, and the interrupt handler can be set to report the error and its cause, or invoke a workaround and continue computing, but posits do *not* make the logical error of assigning a number to something that is, by definition, not a number. This simplifies the hardware considerably. If a programmer finds the need for NaN values, it indicates the program is not yet finished, and the use of *valids* should be invoked as a sort of numerical debugging environment to find and eliminate possible sources of such outputs. Similarly, posits lack a separate ∞ and $-\infty$ like floats have; however, *valids* support open intervals $(maxpos, \infty)$ and $(-\infty, -maxpos)$, which provide the ability to express unbounded results of either sign, so the need for signed infinity is once again an indication that valids are called for instead of posits.

There is no "negative zero" in posit representation; "negative zero" is another defiance of mathematical logic that exists in IEEE floats. With posits, when $a = b$, $f(a) = f(b)$. The IEEE 754 standard says that the reciprocal of "negative zero" is $-\infty$ but the reciprocal of "positive zero" is ∞, but also says that negative zero equals positive zero. Hence, floats imply that $-\infty = \infty$.

Floats have a complicated test for equality, $a \stackrel{?}{=} b$. If either a or b are NaN, the result is always false *even if the bit patterns are identical*. If the bit patterns are *different*, it is still possible for a to equal b, since negative zero equals positive zero! With posits, the equality test is exactly the same as comparing two integers: if the bits are the same, they are equal. If any bits differ, they

are not equal. Posits share the same $a < b$ relation as signed integers; as with signed integers, you have to watch out for wraparound, but you really don't need separate machine instructions for posit comparisons if you already have them for signed integers.

There are no *subnormal* numbers in the posit format, that is, special bit patterns indicating that the hidden bit is `0` instead of `1`. Posits do not use "gradual underflow." Instead, they used tapered precision, which provides the functionality of gradual underflow and a symmetrical counterpart, gradual overflow. (Instead of gradual overflow, floats are asymmetric and use those bit patterns for a vast and unused cornucopia of NaN values.)

Floats have one advantage over posits for the hardware designer: the fixed location of bits for the exponent and the fraction mean they can be decoded in parallel. With posits, there is a little serialization in having to determine the regime bits before the other bits can be decoded. There is a simple workaround for this in a processor design, similar to a trick used to speed the exception handling of floats: Some extra register bits can be attached to each value to save the need for extracting size information when decoding instructions.

3. Bitwise Compatibility and Fused Operations

One reason IEEE floats do not give identical results across systems is because elementary functions like $\log(x)$ and $\cos(x)$ are not required by IEEE to be accurate to the last bit for every possible input. Posit environments must correctly round *all* supported arithmetic operations. (Some math library programmers worry about "The Table-Maker's Dilemma" that some values can take much more work to determine their correct rounding; this can be eliminated by using interpolation tables instead of polynomial approximations.) An incorrect value in the last bit for e^x, say, should be treated the way we would treat a computer system that tells us $2 + 2 = 5$.

The more fundamental reason IEEE floats fail to give repeatable results from system to system is because the standard permits *covert* methods to avoid overflow/underflow and perform operations more accurately, such as by internally carrying extra bits in the exponent or fraction. Posit arithmetic forbids such covert help.

The most recent version (2008) of the IEEE 754 standard [7] includes the fused multiply-add in its requirements. This was a controversial change, and vehemently opposed by many of the committee members. *Fusing* means deferring rounding until the last operation in a computation involving more than one operation, after performing all operations using exact integer arithmetic in a scratch area with a set size. Fusing is not the same as general extended-precision arithmetic, which can increase the size of integers until the computer runs out of memory.

The posit environment **mandates** the following fused operations:

Fused multiply-add	$(a \times b) + c$
Fused add-multiply	$(a + b) \times c$
Fused multiply-multiply-subtract	$(a \times b) - (c \times d)$
Fused sum	$\sum a_i$
Fused dot product (scalar product)	$\sum a_i b_i$

Note that all of the operations in the above list are subsets of the fused dot product [6] in terms of processor hardware requirements. The smallest magnitude nonzero value that can arise in doing a dot product is $minpos^2$. Every product is an integer multiple of $minpos^2$. If we have to perform the dot product of vectors $\{maxpos, minpos\}$ and $\{maxpos, minpos\}$ as an exact operation in a scratch area, we need an integer big enough to hold $maxpos^2/minpos^2$. Recall that

$maxpos = useed^{n-2}$ and $minpos = 1/maxpos$. Hence, $maxpos^2/minpos^2 = useed^{4n-8}$. Allowing for carry bits, and rounding up to a power of 2, tab. 4 shows recommended accumulator sizes.

Table 4. Exact accumulator sizes for each posit size

Posit size in bits, n	16	32	64	128	256
Exact accumulator size, bits	128	1024	4096	65536	1048576

Some accumulators clearly are register-sized data, while the larger accumulators will require a scratch area equivalent of L1 or L2 cache. The fused operations can be done either through software or hardware, but must be available for a posit environment to be complete.

4. Quantitative Comparisons of Number Systems

4.1. A Definition of *Decimal Accuracy*

Accuracy is the inverse of error. If we have a pair of numbers x and y (nonzero and of the same sign), their order-of-magnitude distance is $|\log_{10}(x/y)|$ decades, the same measure that defines the dynamic range with smallest and largest representable positive numbers x and y. A "perfect spacing" of ten numbers between 1 and 10 in a real number system would be not the evenly-spaced counting numbers 1 through 10, but exponentially-spaced $1, 10^{1/10}, 10^{2/10}, \ldots, 10^{9/10}, 10$. That is the *decibel scale*, long used by engineers to define ratios; for example, ten decibels is a factor of 10. Thirty decibels (30 dB) means a factor of $10^3 = 1000$. The ratio 1 dB is about 1.26⋯; if you know a value to within 1 dB, then you have 1 decimal of accuracy. If you knew it to 0.1 dB, that would mean 2 decimals of accuracy, and so on. The formula for *decimal accuracy* is $\log_{10}(1/|\log_{10}(x/y)|) = -\log_{10}(|\log_{10}(x/y)|)$, where x and y are either the correct value and the computed value when using a rounding system like floats and posits, or the lower and upper limits of a bound, if using a rigorous system like intervals or valids.

4.2. Defining Float and Posit Comparison Sets

We can create "scale models" of both floats and posits with only 8 bits each. The advantage of this approach is that the 256 values produced are a small enough set that we can *exhaustively* test and compare all 256^2 entries in the tables for $+ - \times \div$ for numerical properties. A "quarter-precision" float with a sign bit, four-bit exponent field and three-bit fraction field can adhere to all the rules of the IEEE 754 standard. Its smallest positive real (a subnormal float) is 1/1024 and its largest positive real is 240, an asymmetrical dynamic range of about 5.1 decades. Fourteen of the bit patterns represent NaN.

The comparable 8-bit posit environment uses $es = 1$; its positive reals range from 1/4096 to 4096, a symmetrical dynamic range of about 7.2 decades. There are no NaN values. We can plot the decimal accuracy of the positive numbers in both sets, as shown in fig. 7. Note that the values represented by posits represent over two decades more dynamic range than the floats, as well as being as accurate or more for all but the values where floats are close to underflow or overflow. The jaggedness is characteristic of all number systems that approximate a logarithmic representation with piecewise linear sequences. Floats have tapered accuracy only on the left, using gradual underflow; on the right, accuracy falls off a cliff to accommodate all the NaN values. Posits come much closer to a symmetrical tapered accuracy.

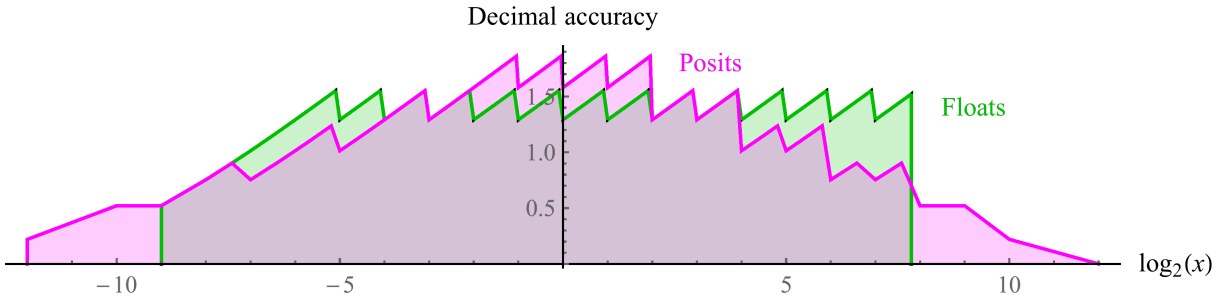


Figure 7. Decimal accuracy comparison of floats and posits

4.3. Single-Argument Operation Comparisons

4.3.1. Reciprocal

For every possible input value x to the function $1/x$, the result can land exactly on another number in the set or it can round, in which case we can measure the decimal loss using the formula of Section 4.1; for floats, it can overflow or produce a NaN. See fig. 8.

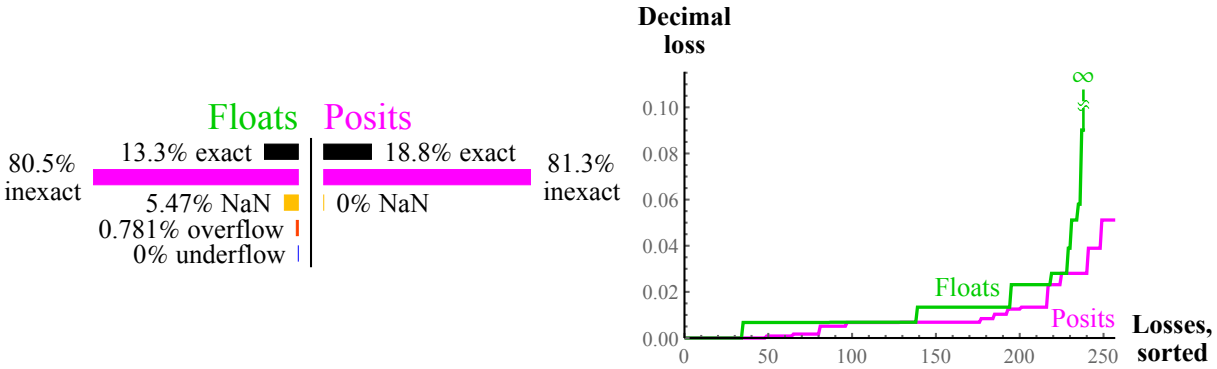


Figure 8. Quantitative comparison of floats and posits computing the reciprocal, $1/x$

The graph on the right plots every decimal accuracy loss created by reciprocation, other than the float cases that produce NaN. Posits are superior to floats, with far more exact cases, and they retain that superiority through their entire range of sorted losses. Subnormal floats overflow under reciprocation, which creates an infinite loss of decimal accuracy, and of course the float NaN *input* values produce NaN output values. Posits are *closed under reciprocation.*

4.3.2. Square Root

The square root function does not underflow or overflow. For negative values, and for the NaN inputs for floats, the result is NaN. Remember that these are "scale model" 8-bit floats and posits; the advantage of posits **increases** with data precision. For a similar plot of 64-bit floats vs. posits, posit error would be about 1/30 of float error instead of about ½ the float error.

4.3.3. Square

Another common unary operation is x^2. Overflow and underflow are a common disaster when squaring floats. For almost half the float cases, squaring does not produce a meaningful numerical value, whereas every posit can be squared to produce another posit. (The square of unsigned infinity is again unsigned infinity.)

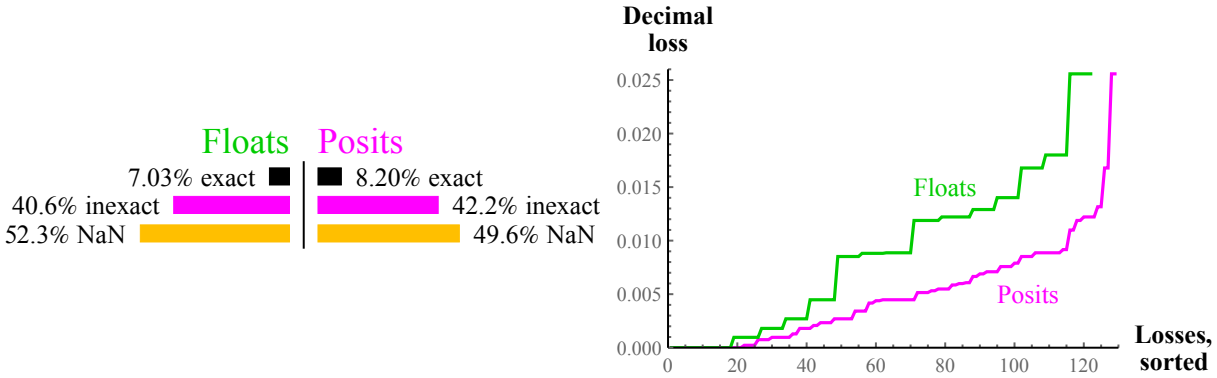


Figure 9. Quantitative comparison of floats and posits computing $\sqrt{x}$

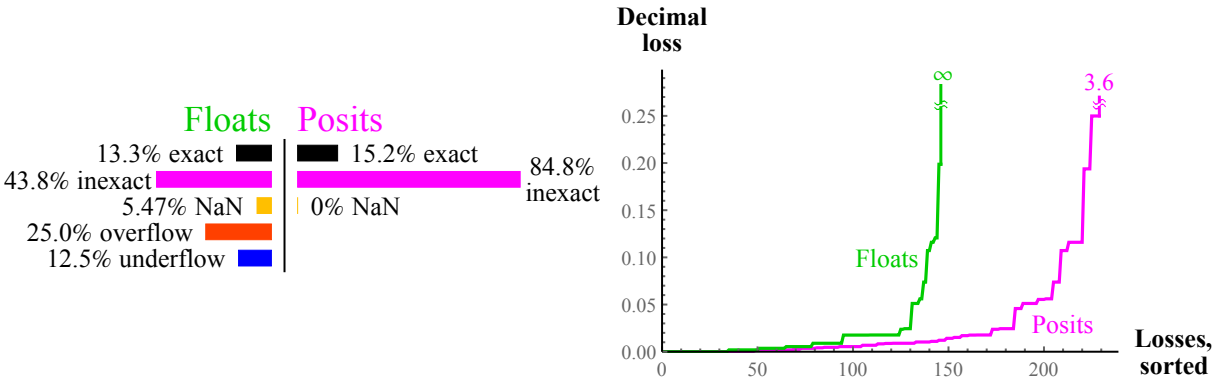


Figure 10. Quantitative comparison of floats and posits computing x^2

4.3.4. Logarithm Base 2

We can also compare logarithm base 2 closure, that is, the percentage of cases where $\log_2(x)$ is exactly representable and how much decimal accuracy is lost when it is not exact. Floats actually have one advantage in that they can represent $\log_2(0)$ as $-\infty$ and $\log_2(\infty)$ as ∞, but this is more than compensated by the richer vocabulary of integer powers of 2 for posits.

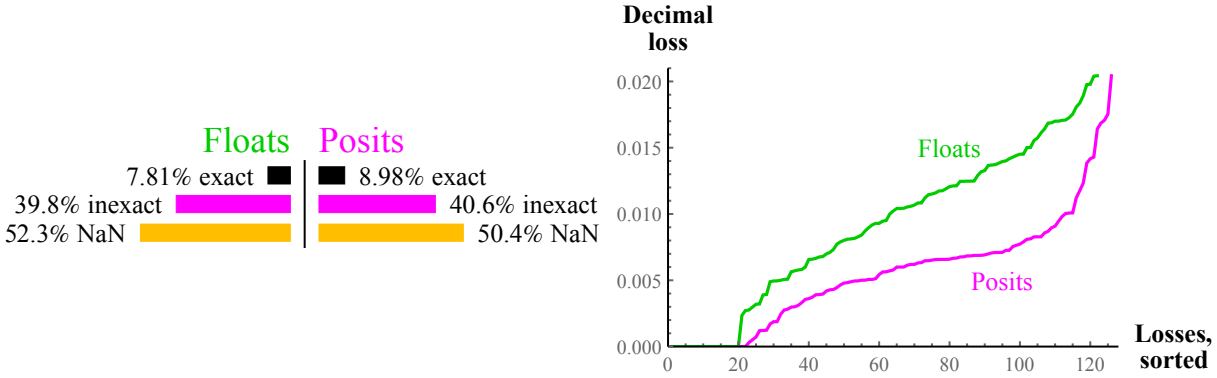


Figure 11. Quantitative comparison of floats and posits computing $\log_2(x)$

The graph is similar to that for square root, with about half the input values resulting in NaN in both cases, but posits experience about half the loss of decimal accuracy. If you can compute $\log_2(x)$, it just takes a scale factor to convert to $\ln(x)$ or $\log_{10}(x)$ or any other logarithm base.

4.3.5. Exponential, 2^x

Similarly, once you can compute 2^x, it is easy to derive a scale factor that also gets you e^x or 10^x and so on. Posits have just one exception case: 2^x is NaN when the argument is $\pm\infty$.

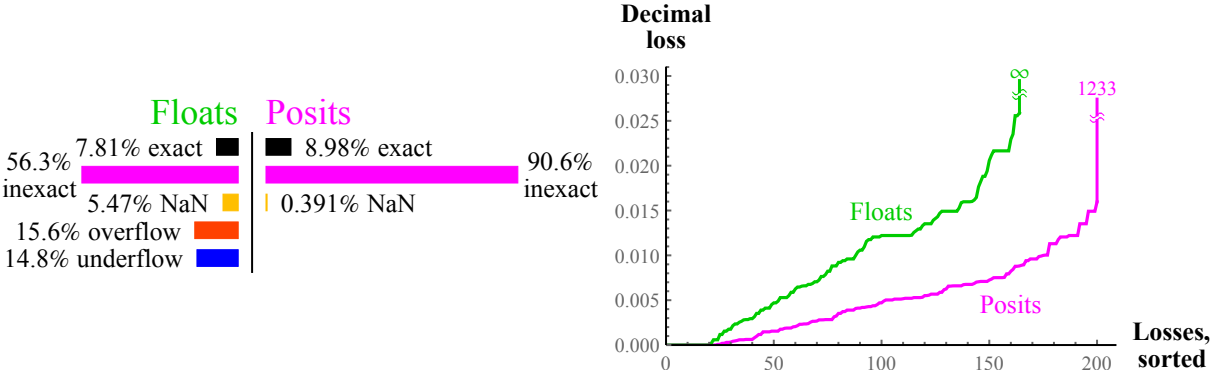


Figure 12. Quantitative comparison of floats and posits computing 2^x

The maximum decimal loss for posits seems large, because 2^{maxpos} will be rounded back to *maxpos*. For this example set, just a few errors are as high as $\log_{10}(24096) \approx 1233$ decimals. Consider which is worse: the loss of over a thousand decimals, or the loss of an *infinite* number of decimals? If you can stay away from those very largest values, posits are still a win, because the error for smaller values is much better behaved. The only time you get a large decimal loss with posits is when working with numbers *far outside of what floats can even express* as input arguments. The graph shows how posits are more robust in terms of the dynamic range for which results are reasonable, and the superior decimal accuracy throughout this range.

For common unary operations $1/x, \sqrt{x}, x^2, \log_2(x)$, and 2^x, posits are consistently and uniformly more accurate than floats with the same number of bits, and produce meaningful results over a larger dynamic range. We now turn our attention to the four elementary arithmetic operations that take two arguments: Addition, subtraction, multiplication, and division.

4.4. Two-Argument Operation Comparisons

We can use the scale-model number systems to examine two-argument arithmetic operations like $+ \; - \; \times \; \div$. To help visualize all 65536 results, we make 256 by 256 "closure plots" that show at a glance what fraction of the results are exact, inexact, overflow, underflow, or NaN.

4.4.1. Addition and Subtraction

Because $x - y = x + (-y)$ works perfectly in both floats and posits, there is no need to study subtraction separately. For the addition operation, we compute $z = x + y$ exactly, and compare it to the sum that is returned by the rules of each number system. It can happen that the result is exact, that it must be rounded to a nearby finite nonzero number, that it can overflow or underflow, or can be an indeterminate form like $\infty - \infty$ that produces a NaN. Each of these is color-coded so we can look at the entire addition table at a glance. In the case of rounded results, the color-coding is a gradient from black (exact) to magenta (maximum error of either posits or floats). fig. 13 shows what the closure plots look like for the floats and the unums.

As with the unary operations, but with far more data points, we can summarize the ability of each number system to produce meaningful and accurate answers:

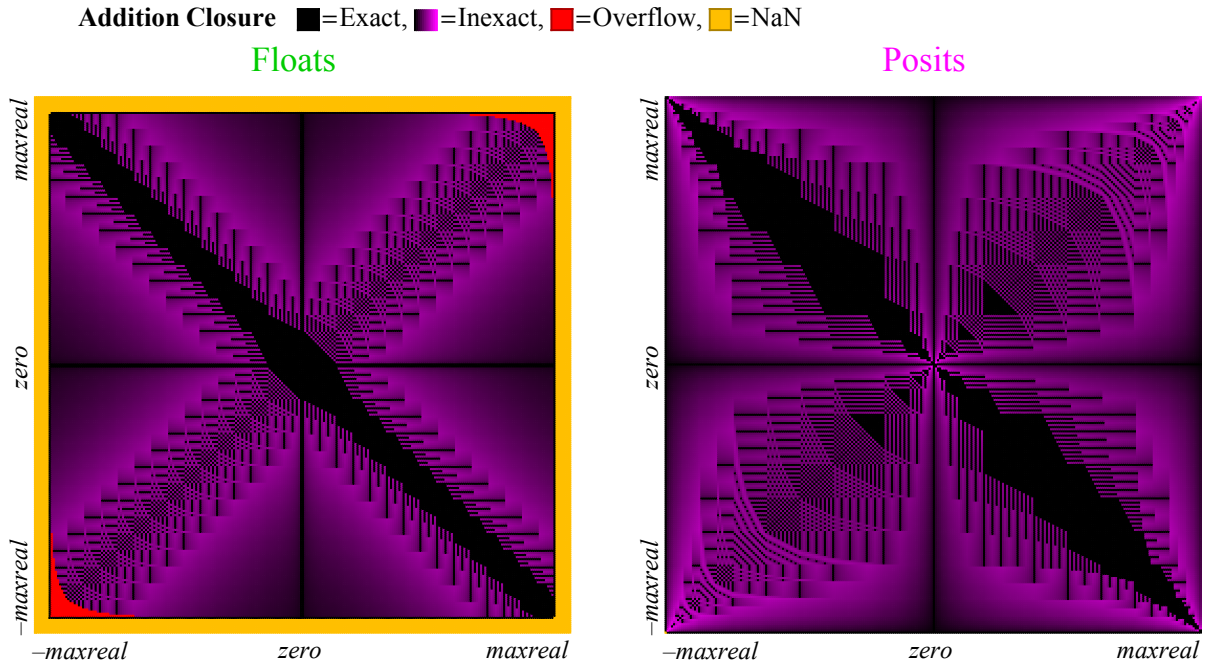


Figure 13. Complete closure plots for float and posit addition tables

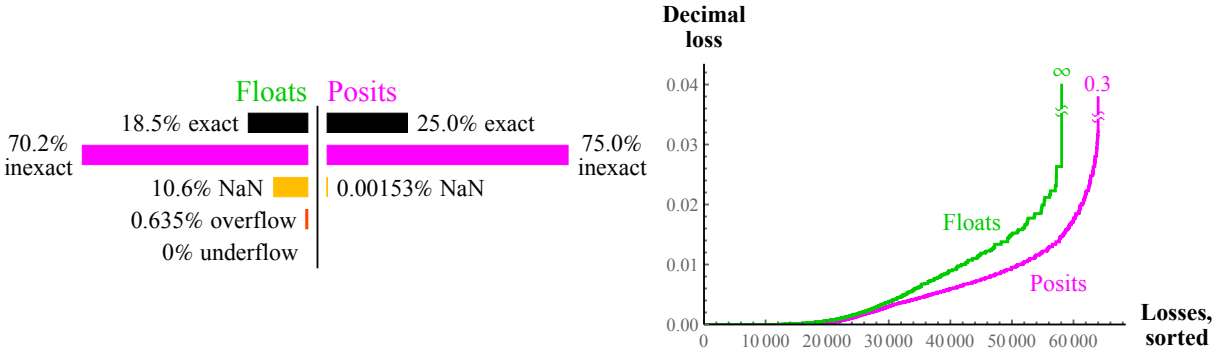


Figure 14. Quantitative comparison of floats and posits for addition

It is obvious at a glance that posits have significantly more additions that are exact. The broad black diagonal band in the float closure plot is much wider than it would be with higher precision, because it represent the gradual underflow region where floats are equally spaced like fixed-point numbers; that is a large fraction of all possible floats when we only have 8 bits.

4.4.2. Multiplication

We use a similar approach to compare how well floats and posits multiply. Unlike addition, multiplication can cause floats to *underflow*. The "gradual underflow" region provides some protection, as you can see in the center of the float closure graph (fig. 15, left). Without it, the blue underflow region would be a full diamond shape. The posit multiplication closure plot is much less colorful, which is a good thing. Only two pixels light up as NaN, near where the axes have their "zero" label. That is where $\pm\infty \times 0$ occurs. Floats have more cases where the product is exact than do posits, *but at a terrible cost*. As fig. 15 shows, almost one-quarter of all float products overflow or underflow, and that fraction does not decrease for higher precision floats.

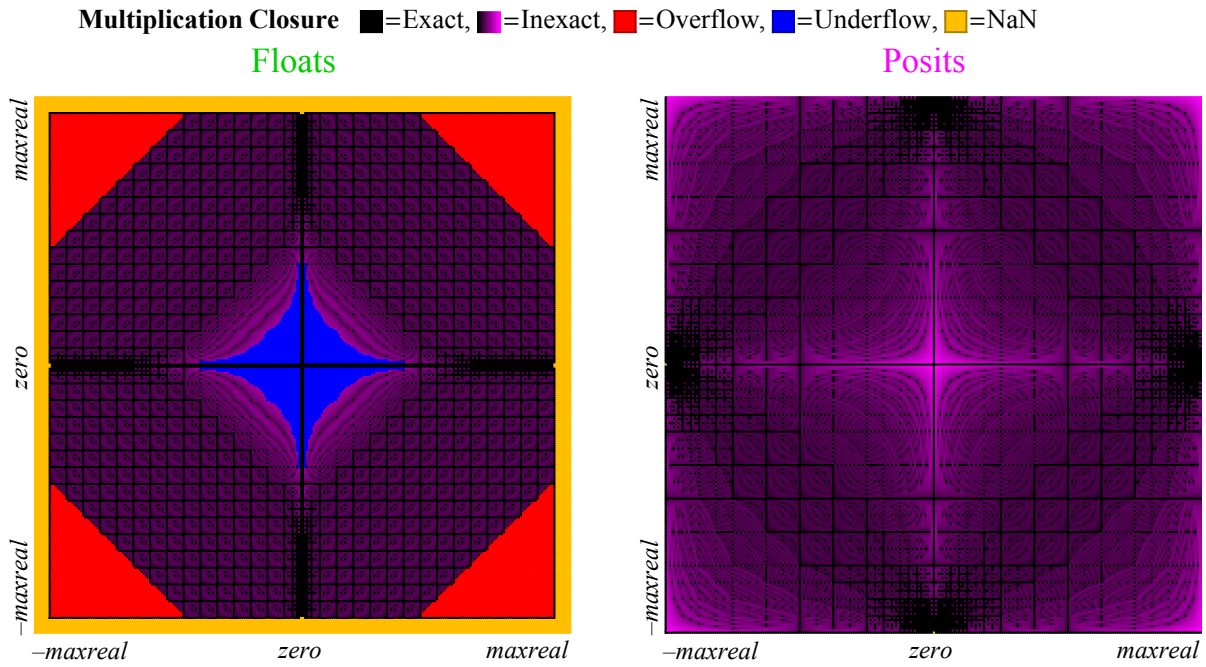


Figure 15. Complete closure plots for float and posit multiplication tables

The worst-case rounding for posits occurs for *maxpos* × *maxpos*, which is rounded back to *maxpos*. For these posits, that represents a (very rare) loss of about 3.6 decimals. As the graph in fig. 16 shows, posits are dramatically better at minimizing multiplication error than floats.

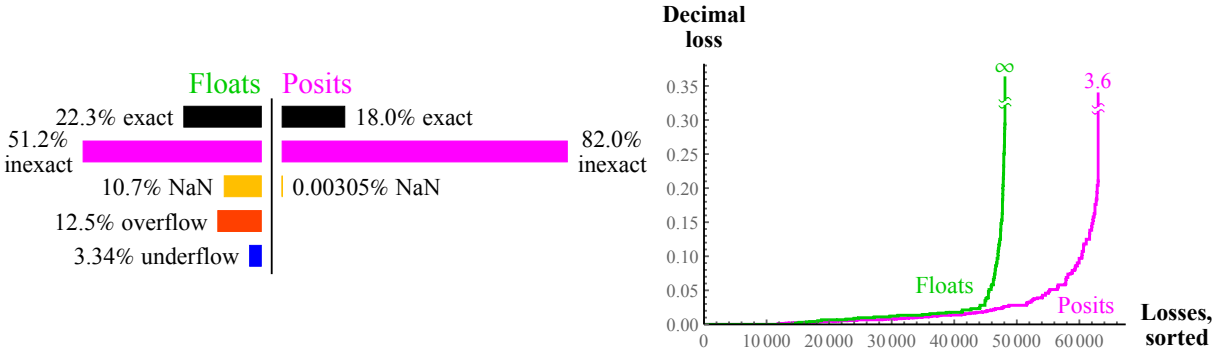


Figure 16. Quantitative comparison of floats and posits for multiplication

The closure plots for the division operation are like those for multiplication, but with the regions permuted; to save space, they will not be shown here. The quantitative comparison for division is almost identical to that for multiplication.

4.5. Comparing Floats and Posits in Evaluating Expressions

4.5.1. The "Accuracy on a 32-Bit Budget" Benchmark

Benchmarks usually make a goal of the smallest execution time, and are frequently vague about how accurate the answer has to be. A different kind of benchmark is one where we fix the precision budget, that is, the number of bits per variable, and try to get the maximum decimal accuracy in the result. Here is an example of an expression we can use to compare various number systems with a 32-bit budget per number:

$$X = \left(\frac{27/10 - e}{\pi - (\sqrt{2} + \sqrt{3})} \right)^{67/16} = 302.8827196\cdots$$

The rule is that we start with the best representation the number system has for e and π, and for all the integers shown, and see how many decimals agree with the correct value for X after performing the nine operations in the expression. We will show incorrect digits in orange.

While IEEE 32-bit floats have decimal accuracy that wobbles between 7.3 and 7.6 decimals, the accumulation of rounding error in evaluating X gives the answer 302.912···, with only three correct digits. This is one reason IEEE float users feel pressure to use 64-bit floats everywhere, because even simple expressions risk losing so much accuracy that the results might be useless.

The 32-bit posits have tapered decimal accuracy, with a wobble between about 8.2 and 8.5 decimals for numbers near 1 in magnitude. In evaluating X, they give the answer 302.88231···, twice as many significant digits. Bear in mind that 32-bit posits have a dynamic range of over 144 decades, whereas 32-bit IEEE floats have a much smaller dynamic range of about 83 decades. Therefore, the extra accuracy in the result was *not* attained at the expense of dynamic range.

4.5.2. A Quad-Precision Test: Goldberg's Thin Triangle Problem

Here's a classic "thin triangle" problem [1]: Find the area of a triangle with sides a, b, c, when two of the sides b and c are just 3 Units in the Last Place (ULPs) longer than half the longest side a (fig. 17):

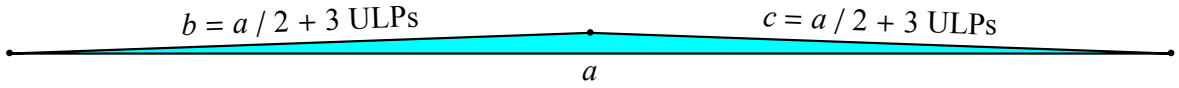


Figure 17. Goldberg's thin triangle problem

The classic formula for the area A uses a temporary value s:

$$s = \frac{a+b+c}{2}; A = \sqrt{s(s-a)(s-b)(s-c)}$$

The hazard in the formula for a thin triangle is that s is very close to the value of a, so the calculation of $(s - a)$ magnifies any rounding into a large relative error. Let's try 128-bit (quad-precision) IEEE floats, where $a = 7, b = c = 7/2 + 3 \times 2^{-111}$. (If the units are in light-years, then the short sides are only longer than half the long side by 1/200 the diameter of a proton. Yet that pops the triangle up to about the width of a doorway at the thickest point.) We also evaluate the formula for A using 128-bit posits ($es = 7$). Here are the results:

Correct answer:	$3.14784204874900425235885265494550774498\cdots \times 10^{-16}$
128-bit IEEE float answer:	$3.63481490842332134725920516158057682788\cdots \times 10^{-16}$
128-bit posit answer:	$3.14784204874900425235885265494550774439\cdots \times 10^{-16}$

Posits have as much as 1.8 decimal digits more accuracy than floats in quad precision over a vast dynamic range: from 2×10^{-270} to 5×10^{-269}. That is ample for protecting against this particular inaccuracy-amplifying disaster. It is interesting to note that the posit answer would be more accurate than the float answer even if it were converted to a 16-bit posit at the end.

4.5.3. The Quadratic Formula

There is a classic numerical analysis "trick" to avoid a rounding error hazard in finding the roots r_1, r_2 of $ax^2 + bx + c = 0$ using the usual formula $r_1, r_2 = (-b \pm \sqrt{b^2 - 4ac})/(2a)$ when b is much larger than a and c, resulting in left-digit cancellation for the root for which $\sqrt{b^2 - 4ac}$ is very close to b. Instead of expecting programmers to memorize a panoply of arcane tricks for every formula, perhaps posits can make it a bit safer to simply apply the textbook form of the formula. Suppose $a = 3, b = 100, c = 2$, and compare 32-bit IEEE floats with 32-bit posits.

Table 5. Quadratic equation evaluation

Root	Mathematical Value	Float result	Posit result
r_1	$(-100 + \sqrt{9976})/6 = -0.0200120144\cdots$	-0.02001190	$-0.0200120\mathbf{6}\cdots$
r_2	$(-100 - \sqrt{9976})/6 = -33.3133213\cdots$	-33.3133302	$-33.313321\mathbf{6}\cdots$

The numerically unstable root is r_1, but notice that 32-bit posits still return 6 correct decimals instead of only 4 correct decimals from the float calculation.

4.6. Comparing Floats and Posits for the Classic LINPACK Benchmark

The basis for ranking supercomputers has long been the time to solve an n-by-n system of linear equations $\mathbf{A}x = b$. Specifically, the benchmark populates a matrix $\mathbf{A}$ with pseudo-random numbers between 0 and 1, and sets the b vector to the row sums of $\mathbf{A}$. That means the solution x should be a vector of all 1s. The benchmark driver calculates the norm of the residual $||\mathbf{A}x - b||$ to check correctness, though there is no "hard number" in the rules that limits how inaccurate the answer can be. It is typical to lose several decimals of accuracy in the calculation, which is why it requires 64-bit floats to run (not necessarily IEEE). The original benchmark size was $n = 100$, but that size became too small for the fastest supercomputers so n was increased to 300, and then to 1000, and finally (at the urging of the first author) changed to a *scalable* benchmark that ranks based on operations per second assuming $^2/_3n^3 + 2n^2$ multiply or add operations.

In comparing floats and posits, we noticed a subtle flaw in the benchmark driver: The answer is generally *not* a sequence of all 1 values, because *rounding occurs when computing the row sums*. That error can be eliminated by finding which entries in $\mathbf{A}$ contribute a **1** bit to the sum beyond the precision capabilities, and setting that bit to **0**. This assures that the row sum of $\mathbf{A}$ is representable without rounding, and then the answer x actually **is** a vector of all 1s.

For the original 100-by-100 problem, 64-bit IEEE floats produce an answer that looks like

0.9999999999999633626401873698341660201549530029296875
1.0000000000000011102230246251565404236316680908203125
⋮
1.0000000000000022648549702353193424642086029052734375

Not a single one of the 100 entries is correct; they are close to 1, but never land on it. With posits, we can do a remarkable thing. Use *32-bit* posits and the same algorithm; calculate the residual, $r = \mathbf{A}x - b$, using the fused dot product. Then solve $\mathbf{A}x' = r$ (using the already-factored $\mathbf{A}$) and use x'as a correction: $x \leftarrow x - x'$. The result is an unprecedented **exact** answer to the LINPACK benchmark: $\{1, 1, \ldots, 1\}$. Can the rules of LINPACK forbid the use of a new 32-bit number type that can get the perfect result with a residual of *zero*, and continue to insist on

the use of 64-bit float representations that cannot? That decision will be up to the shepherds of the benchmark. For those who use linear equations to solve real problems and not to compare supercomputer speeds, posits offer an overwhelming advantage.

5. Summary

Posits beat floats at their own game: guessing their way through a calculation while incurring rounding errors. Posits have higher accuracy, larger dynamic range, and better closure. They can be used to produce better answers with the same number of bits as floats, or (what may be even more compelling) an equally good answer with *fewer bits*. Since current systems are bandwidth-limited, using smaller operands means higher speed and less power use.

Because they work like floats, not intervals, they can be regarded as a drop-in replacement for floats, as demonstrated here. If an algorithm has survived the test of time as stable and "good enough" using floats, then it will run even better with posits. The fused operations available in posits provide a powerful way to prevent rounding error from accumulating, and in some cases may allow us to safely use 32-bit posits instead of 64-bit floats in high-performance computing. Doing so will generally increase the speed of a calculation $2-4\times$, and save energy and power as well as the cost of storage. Hardware support for posits will give us the equivalent of one or two turns of Moore's law improvement, without any need to reduce transistor size and cost.

Unlike floats, posits produce bitwise-reproducible answers across computing systems, overcoming the primary failure of the IEEE 754 float definition. The simpler and more elegant design of posits compared to floats reduces the circuitry needed for fast hardware. As ubiquitous as floats are today, posits could soon prove them obsolete. □

References

1. David Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys (CSUR)*, 23(1):5–48, 1991. DOI:10.1145/103162.10316.

2. John L Gustafson. *The End of Error: Unum Computing*, volume 24. CRC Press, 2015.

3. John L Gustafson. Beyond Floating Point: Next Generation Computer Arithmetic. Stanford Seminar: `https://www.youtube.com/watch?v=aP0Y1uAA-2Y`, 2016. full transcription available at `http://www.johngustafson.net/pdfs/DebateTranscription.pdf`.

4. John L Gustafson. A radical approach to computation with real numbers. *Supercomputing Frontiers and Innovations*, 3(2):38–53, 2016. DOI:10.14529/jsfi160203.

5. John L Gustafson. The Great Debate @ ARITH23. `https://www.youtube.com/watch?v=KEAKYDyUua4`, 2016. full transcription available at `http://www.johngustafson.net/pdfs/DebateTranscription.pdf`.

6. Ulrich W Kulisch and Willard L Miranker. *A new approach to scientific computation*, volume 7. Elsevier, 2014.

7. More Sites. IEEE standard for floating-point arithmetic. *IEEE Computer Society*, 2008. DOI:10.1109/IEEESTD.2008.4610935.

8. Isaac Yonemoto. `https://github.com/interplanetary-robot/SigmoidNumbers`.

DOI: 10.14529/jsfi170207

InfiniCortex - From Proof-of-concept to Production

Gabriel Noaje[1], *Alan Davis*[1], *Jonathan Low*[1], *Lim Seng*[1], *Tan Geok Lian*[1], *Łukasz P. Orłowski*[1,2,3], *Dominic Chien*[1], *Liou Sing-Wu*[1], *Tan Tin Wee*[1,4,5], *Yves Poppe*[1], *Ban Hon Kim Kenneth*[6], *Andrew Howard*[7], *David Southwell*[8], *Jason Gunthorpe*[8], *Marek T. Michalewicz*[1,2,9]

The global effort to build ever more powerful supercomputers is faced with the challenge of ramping up High Performance Computing systems to ExaScale capabilities and, at the same time, keeping the electrical power consumption for a system of that scale at less than 20 MW level. One possible solution, bypassing this local energy limit, is to use distributed supercomputers to alleviate intense power requirements at any single location. The other critical challenge faced by the global computer industry and international scientific collaborations is the requirement of streaming colossal amounts of time-critical data. Examples abound: i) transfer of astrophysical data collected by the Square Kilometre Array to the international partners, ii) streaming of large facilities experimental data through the Pacific Research Platform collaboration of DoE, ESnet and other partners in the US and elsewhere, iii) the Superficilities vision expressed by DoE, iv) new architecture for CERN LHC data processing pipeline focussing on more powerful processing facilities connected by higher throughput connectivity.

The InfiniCortex project led by A*STAR Computational Resource Centre demonstrates a worldwide InfiniBand fabric circumnavigating the globe and bringing together, as one concurrent globally distributer HPC system, several supercomputing facilities spanned across four continents (Asia, Australia, Europe and North America). Using global scale InfiniBand connections, with bandwidth utilisation approaching 98% link capacity, we have established a new architectural approach which might lead to the next generation supercomputing systems capable of solving the most complex problems through the aggregation and parallelisation of many globally distributed supercomputers into a single hive-mind of enormous scale.

Keywords: InfiniCortex, InfiniBand, global supercomputer connectivity, superfacilities, InfiniCloud, HPC Cloud, supercomputer networking, HPC workflows, ADIOS.

Introduction

This article is a final report of *InfiniCortex I* project led by A*STAR Computational Resource Centre in Singapore. We document an implementation of a worldwide InfiniBand fabric bringing together several supercomputing facilities spanned across the globe to create a galaxy of supercomputers [12]. *InfiniCortex I* project represents a huge collaboration effort of several agencies and universities in Singapore (A*STAR, NSCC, NUS, NTU, SingAREN) together with more than 20 international partners around the globe.

After successfully demonstrating the first 100Gbps transcontinental InfiniBand connection connecting Singapore and United States of America at the annual Supercomputing Conference

[1]A*STAR Computational Resource Centre (A*CRC), Singapore
[2]Institute for Advanced Computational Science, Stony Brook University, New York, USA
[3]Department of Applied Mathematics and Statistics, Stony Brook University, New York, USA
[4]Singapore National Supercomputing Centre (NSCC), Singapore
[5]Department of Biochemistry, Yong Loo Lin School of Medicine, National University of Singapore, Singapore
[6]National University of Singapore, Singapore
[7]The Australian National University, Canberra, Australia
[8]Obsidian Strategics Inc, Edmonton, Canada
[9]Interdisciplinary Centre for Mathematical and Computational Modelling (ICM), University of Warsaw, Warsaw, Poland

2014, in New Orleans, USA [10], the award winning InfiniCortex project [1] grew rapidly demonstrating every year novel capabilities.

Over the last few years several unprecedented elements have been showcased:

- largest ever spanning InfiniBand network – a ring-around-the-world with most of the segments at 100Gbps and few at 10-30Gbps;
- eight InfiniBand subnets created using InfiniBand routers and demonstrated InfiniBand routing using BGFC (Bowman Global Fabric Controllers [5]);
- InfiniCloud: the first ever true high throughput global span HPC cloud allowing instances provisioning across four continents: Asia, Australia, North America and Europe [7–9].

During the last three years, InfiniCortex and numerous applications utilising this concept and infrastructure, have been successfully demonstrated at several international events: Supercomputing Frontiers 2015 and 2016 in Singapore; ISC15 and ISC16 in Frankfurt, Germany; TNC15 in Porto, Portugal and TNC16 in Prague, Czech Republic and finally at SC14 (New Orleans) and SC15 (Austin), USA.

Hence we have furnished a sufficient proof of concept demonstrations exhibiting the effectiveness of the proposed solutions. Several elements are already being implemented in Singapore and elsewhere as production solutions enabling higher bandwidth and security.

In the next section we will describe in some detail the third stage on our *InfiniCortex I* project which took place during the Supercomputing 2016 conference in Salt Lake City, USA. In Sub-Section 1.1 we will provide a list of all our collaborators in this project, followed in Sub-Section 1.2 with a description of a global scale network infrastructure, and, in Section 1.3, details of a number of application demonstrations prepared with our partners from the Oak Ridge National Laboratory, Fermilab, Stony Brook University, George Washington University, USA; University of Reims Champagne-Ardenne, France; Poznań Supercomputing and Network Centre and Interdisciplinary Centre for Mathematical and Computational Modelling, University of Warsaw, Poland. In Section 2 we online our plans for InfiniCortex 2 phase of our project, and finally Section 3 contains conclusions of this report.

1. InfiniCortex Demonstrations at SuperComputing 2016

The *International Conference for High Performance Computing, Networking, Storage and Analysis - SC16*, the 28th annual international conference of high performance computing, networking, storage and analysis, celebrated the contributions of researchers and scientists – from those just starting their careers to those whose contributions have made lasting impacts. The conference drew more than 11,100 registered attendees and featured a technical program spanning six days. The exhibit hall featured 349 exhibitors from industry, academia and research organizations from around the world.

During the conference, Salt Lake City also became the hub for the world's fastest computer network: SCinet, SC16's custom-built network which delivered 3.15 terabits per second in bandwidth. The network featured 56 miles of fiber deployed throughout the convention center and 32 million dollars in loaned equipment. InfiniCortex is build on top of the SCinet network with support from the SCinet team and in collaboration with various netowrking orgnizations.

1.1. Partners

The following partners were involved in the InfiniCortex demonstrations at SC16:

- A*STAR Computational Resource Centre* - Singapore
- Oak Ridge National Laboratory (ORNL) - USA
- Fermilab - USA
- Stony Brook University (SBU)* - USA
- George Washington University (GWU)* - USA
- University of Reims Champagne-Ardenne (URCA)* - France
- Poznań Supercomputing and Network Centre (PSNC)* - Poland
- Interdisciplinary Centre for Mathematical and Computational Modelling (ICM)* - Poland

Locations marked with an asterisk denote locations where a Longbow InfiniBand range extenders were installed for the SC16 demo.

1.2. Network Infrastructure

The InfiniBand network ran on top of the dark-fibre network infrastructure prepared by A*CRC Network team in collaboration with various networking organisations (SingAREN, TEIN, GEANT, PIONEER, RENATER, Internet2, SCinet). A total of five E100 Longbows were used to connect the SC16 show floor to A*CRC in Singapore providing a 50Gbps InfiniBand link.

A global WAN InfiniBand network has been setup with 4 distinct subnets:

- National Supercomputer Centre, Singapore
- Interdisciplinary Centre for Mathematical and Computational Modelling (ICM), Poland
- A*CRC, Singapore + URCA, France + George Washington University, USA
- Stony Brook University, USA

using Obsidian's BGFC InfiniBand subnet manager [5] capable of InfiniBand routing between subnets.

Figure 1. SC16 InfiniCortex global coverage map

1.2.1. Performance metrics

InfiniBand

- Each Longbow E100 link is capable of 10Gbps of InfiniBand traffic. All 5 usable Longbows were stress-tested for bandwidth. All five Longbows were pumping 10Gbps simultaneously - giving 50Gbps raw bandwidth (40Gbps usable data bandwidth due to 2 in every 10 bits for encoding).
- The bandwidth test of a single Longbow shows 938.83 MB/s between a server at SC16 and a remote server at A*CRC in Singapore.

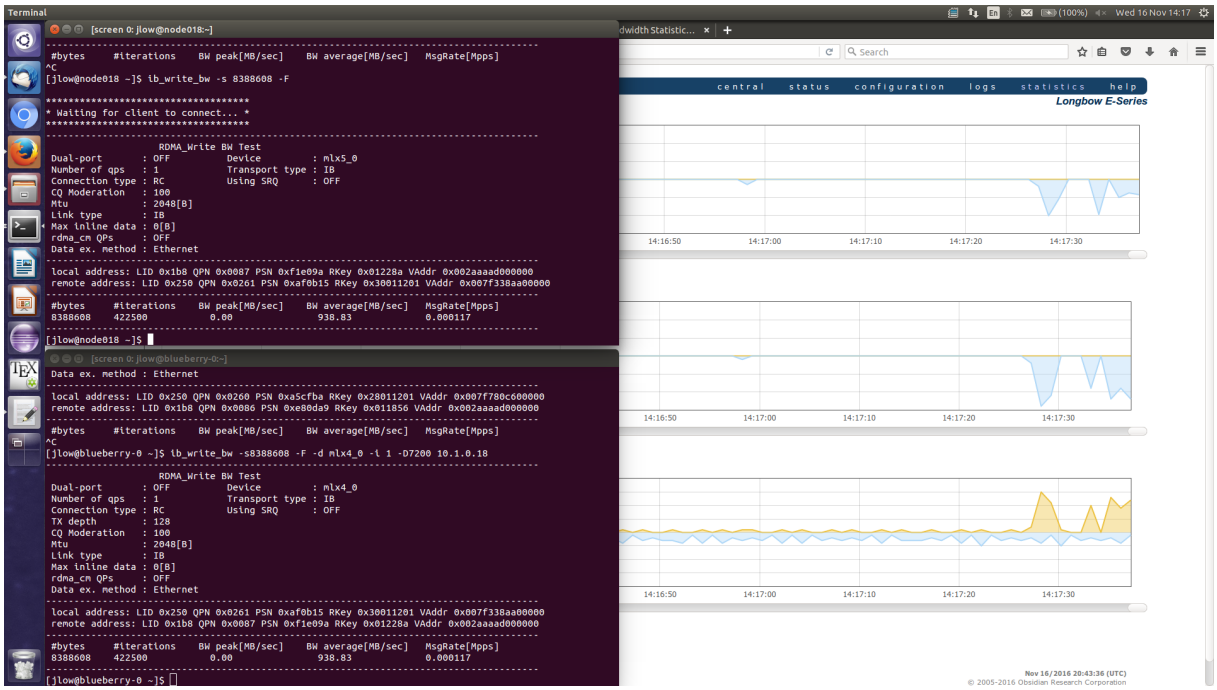


Figure 2. Global RDMA test between Singapore and Salt Lake City using Longbows E100

- An aggregated total bandwidth of 75 - 80 Gbps was utilised with link sharing with Tokyo-Tech University

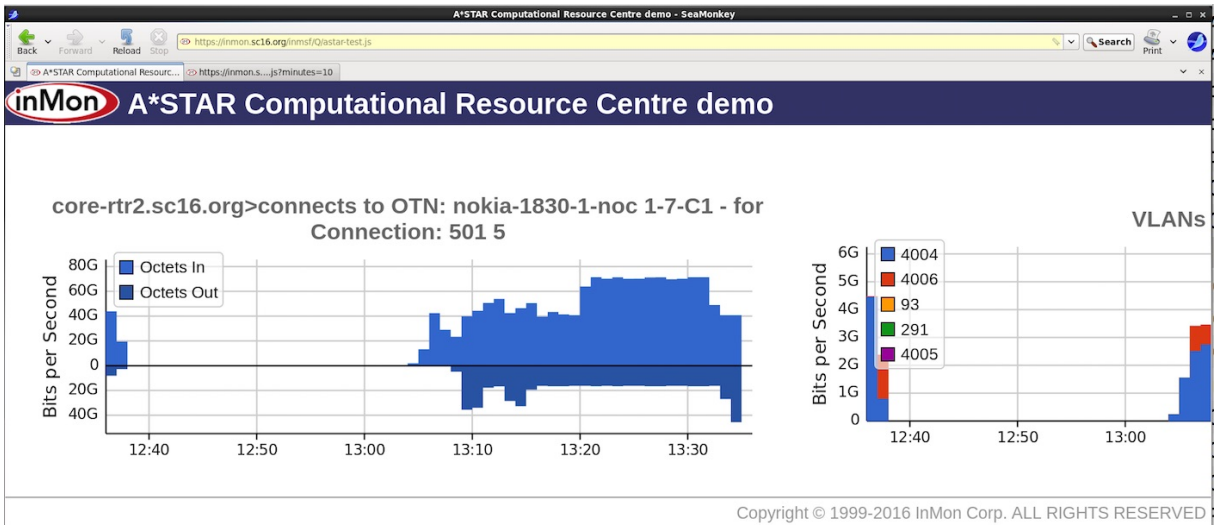


Figure 3. Screen-shot of the bandwidth utilisation during SC16 demos showing 75-80 Gbps data transfer between Tokyo University of Technology and A*CRC booth at the show floor

- The whole SC16 exhibit utilised just over 800 GBps bandwidth.
- A dsync+ test was attempted to transfer a large dataset from storage in A*CRC to SC16. The initial transfer was 6MB/s due to heavy packet loss on the link - despite no recorded packet loss during the bandwidth stress tests. There was no time left to diagnose the issue.

100G Ethernet

Andrew Howard from NSCC, Canberra, Australia conducted the following additional tests:

- iperf3 network test showed 16-23Gbps per UDP stream, 17.2Gbps for TCP.

- Lim Seng from A*CRC did further Ethernet bandwidth tests and was able to add additional bandwidth to the network.

1.3. Demonstrations

During SC16 A*CRC and several partners demonstrated five applications that were using the long range InfiniBand connectivity. The following sections will briefly describe the details and achievements of each applications.

1.3.1. Demonstrations with Oak Ridge National Laboratory (ORNL) / Fermilab / Stony Brook University (SBU)

The demonstration with ORNL and SBU called *Remote Fusion Experiment Data Analysis Through Wide-Area Network* consisted in remote data processing capability of large and high-throughput science experiment through cross-Pacific wide area networks and showed how one can manage science workflow executions remotely by using ORNL ADIOS data management system and FNAL mdtmFTP data transfer system. In this demonstration our partners presented a fusion data processing workflow, called Gas Puff Imaging (GPI) analysis, to detect and trace blob movements during fusion experiment. GPI data streams were being sent from Singapore to Fermilab for near-real time analysis, while ADIOS was managing analysis workflows and mdtmFTP transports stream data.

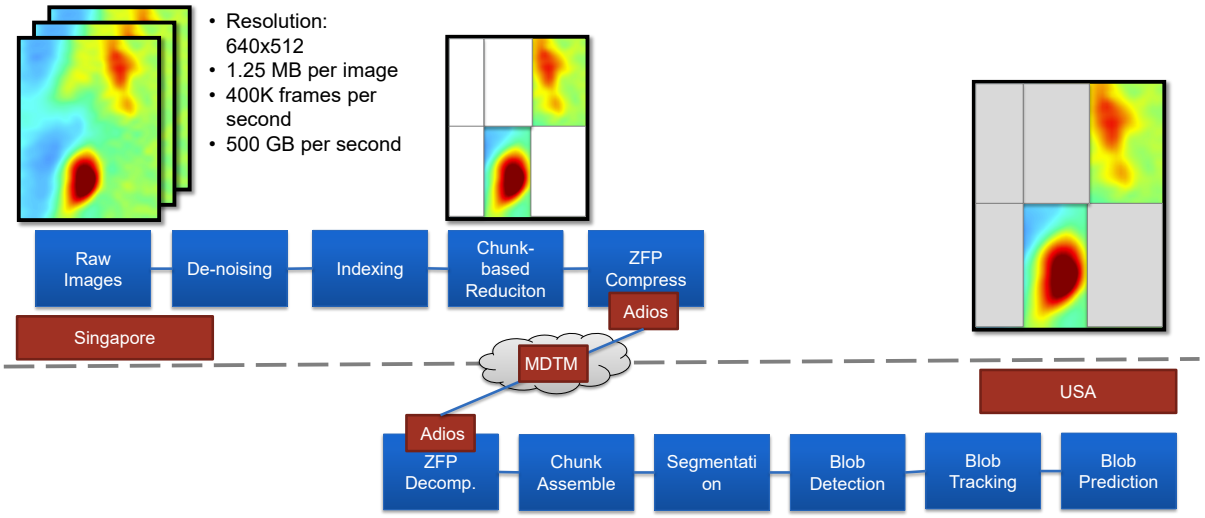


Figure 4. NSTX GPI Workflow With ADIOS and MDTM

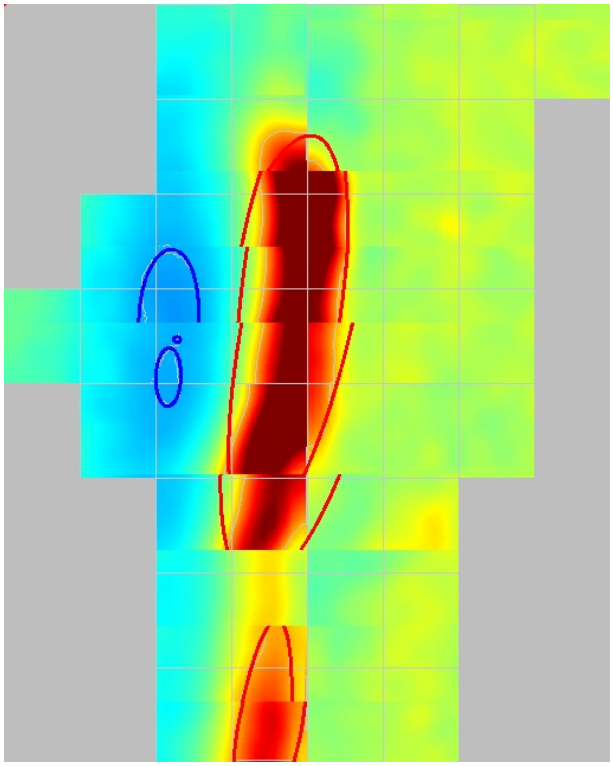


Figure 5. ORNL demonstration screenshot

Figure 5 shows a screenshot of the live demo running in SC16. The demonstration was susscesfully relying on the mdtmFTP data transfer system developed in Fermilab.

Accomplishments and problems encountered:

We encountered several problems with this demo especially because the Stony Brook servers were available only a few days before SC16. Most of the tests have been done between Singapore and Fermilab, however even in this scenario a lot of problems arose from the fact this was not a dedicated L2 circuit and several firewalls were blocking the communication on each side.

Ultimately the problems have been solved. ORNL team have had plans to show this demo once again in 2017. The demo is part of a bigger collaboration between ORNL and Japan in the ITER project.

1.3.2. Demonstrations with George Washington University (GWU)

A Preliminary Study of Executing Parallel Applications over a Long-Range-IB network was showcased using mpiBLAST - a freely available, open-source, parallel implementation of NCBI BLAST. The mpiblast was run on multiple nodes on a cluster comprising of 4 nodes at GWU and 3 nodes at A*CRC and communicating was done via mpi/LHIB.

The experiment consisted of the following steps:

- A protein reference database (524603 protein entries, size of 153MB) was prepared and distributed to all of the compute nodes of the cluster (in both USA and Singapore).
- The database was fragmented into 64 smaller fragments by running the program: mpiformatdb.
- A subset of the protein sequences (from the reference database) was used for the protein blast search (blastp). The total number of sequences used was 7516.

- The total execution time of the blastp (protein blast search) was measured against the different number of compute nodes used. Measurements were conducted using two cases: i) case1: compute nodes on USA only and ii) case2: compute nodes in both USA and Singapore.
- The mpiblast command is as follows:
  `mpirun -np 9 -mca btl openib,self -hostfile hostfile mpiblast -copy-via=cp /-concurrent=8 -use-parallel-write -query-segment-size=1000 -p blastp /-d AR.faa -i testInput.faa -o testOutput.txt`
  where:
  –np changing from 9, 17, 33 (for 1 node, 2 nodes and 4 nodes)
  –concurrent changing from 8, 16, 32 (for 1 node, 2 nodes and 4 nodes)
  –copy-via specify to use system cp command to copy fragment database files onto nodes.
  –mca btl specifies to use openib as the communication protocol
  –hostfile specifies the host machines being used, for example:
  10.1.1.30 (node at USA)
  10.1.1.20 (node at Singapore)
  –AR.faa is the protein reference database
  –testInput.faa is the protein blast input sequence file
  –segment-size specifies the job size (no. of sequence send from master mpi process to the workers mpi processes to work; i.e. for controlling task granularity)

Results of the tests are shown in figures 6 and 7.

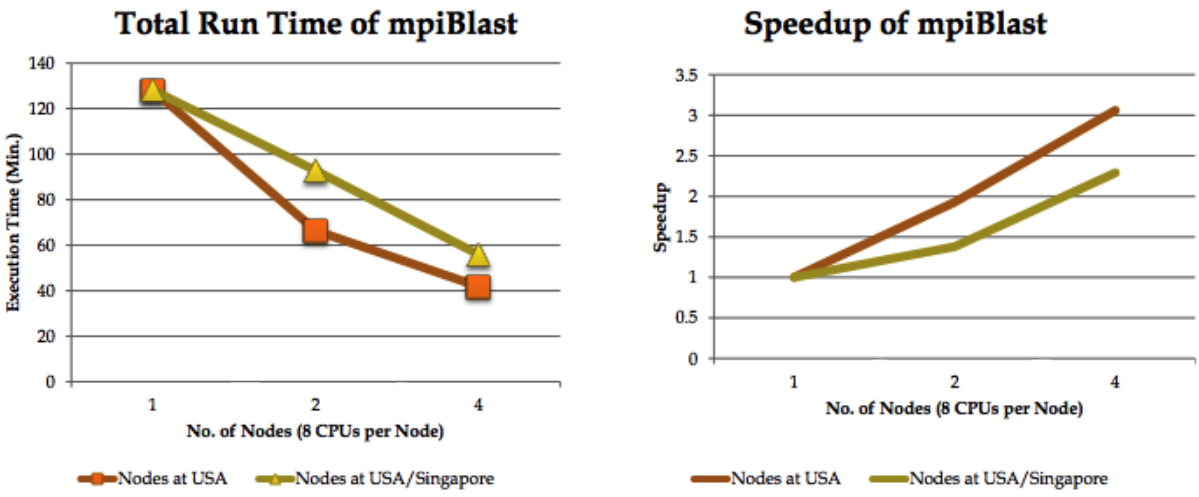


Figure 6. Input sequences: 7516

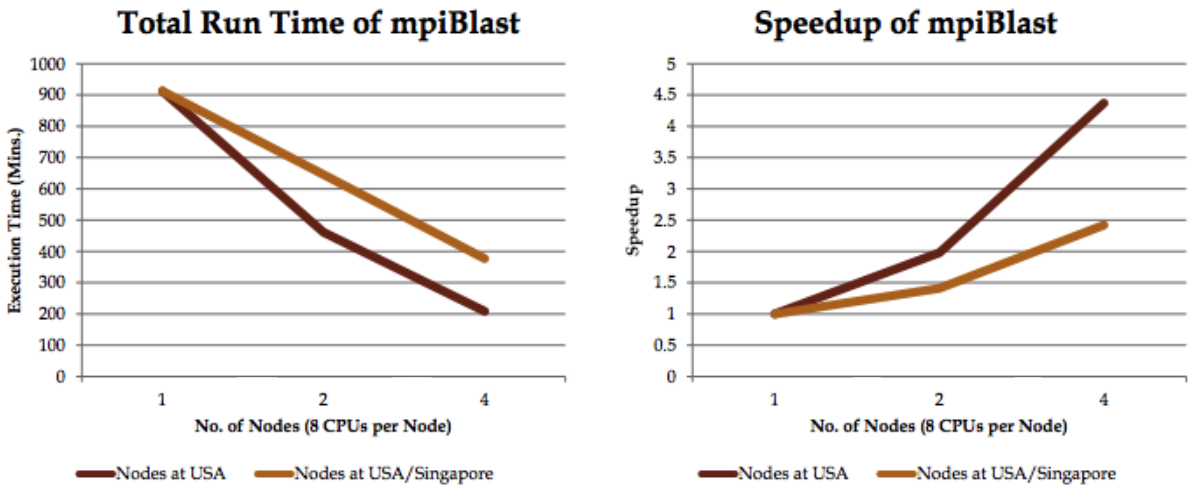


Figure 7. Input sequences: 48844

The following conclusions were drawn after analizing the results:

- Up to 32 individual tasks were running on different nodes. In figures 6 and 7, the computational scalability is shown with different problem sizes using different numbers of computing nodes, and also reveal the difference of localising the process on one side versus distributed process across InfiniCortex infrastructure.
- Super linear scalability is observed for the processes running only on the US nodes. Super linear scalability is unusual and there is no guarantee if cluster size is expanded. The main reason behind this super linear scalability is unclear yet, but it may be because the overhead of the initial data distribution is smaller amongst the nodes connected within the same InfiniBand switch.
- Basically, a linear scalability for the task distributed across InfiniCotex is observed, and it is the ideal case for a parallel computational process. So it should be considered as a successful demonstration.
- Many large scale scientific and engineering computational tasks can be divided into many small sub-tasks using data partitioning strategy, and then computed in parallel using MPI (Message Passing Interface) protocol to distribute tasks and data on different computer nodes. However, the efficiency of the rapid data exchange amongst the sub-task is very sensitive to the network latency, and thus certain computational tasks are inherently not scalable on the InfiniCortex infrastructure.
- To hide the inevitable latency due to the distance with a large data transfer, we have successfully demonstrated a number of workflows since SC14 (i.e. pipelining different stages of a task on the systems in different locations), but this is the first time we demonstrate solving a single computational task on two HPC clusters across continents using MPI. This specific application was succesfully run because it is inherently embarrassingly parallel, no data exchange required among the sub-tasks.
- Despite the success of this demonstration, we have encountered several difficulties:
  - OpenMPI was unable to build on the cluster on US side, and the issue was eventually resolved by the A*CRC software team. It was mainly due to the version of OpenMPI being too new for the building scripts that were used.

– We observed a big overhead (up to 2 minutes) for the initialisation of MPI if the job is distributed across InfiniCortex. It was confirmed that this overhead does not affect the accuracy of the computation, but the cause is not clear yet.
– Because the servers in the US and Singapore were using different types of processors additional time was necessary for tuning the data set between the two clusters. Such heterogeneity made difficult a fair comparison.

1.3.3. Demonstrations with University of Reims Champagne-Ardenne (URCA)

rCUDA (`http://www.rcuda.net`) is a development of the Parallel Architectures Group from Universitat Politecnica de Valencia (Spain). rCUDA enables the concurrent remote usage of CUDA-enabled devices in a transparent way. Thus, the source code of applications does not need to be modified in order to use remote GPUs but rCUDA takes care of all the necessary details. Furthermore, the overhead introduced by using a remote GPU is very small.

ROMEO HPC Center succesfully tested and run rCUDA inside their 260 GPUs cluster for the past year. The purpose of the SC16 demonstration was to test rCUDA over InfiniCortex and analyse the behaviour of the framework over extremely long distances. For the demo purpose the rCUDA server was installed in Singapore on a machine that didn't have any GPUs attached. The rCUDA client was installed on 18 servers in Reims each having 2 K20x. A standard matrix-matrix multiplication example from the CUDA SDK was then run on the Singapore machine which transparently sent all CUDA calls to the servers in Reims where the computation was actually taking place on the GPUs and then it was collecting the final result.

The rCUDA developers from Universitat Politecnica de Valencia developed a small graphical interface that was showing the performance of one single node compared to the performance of running the same code over several nodes. This was clearly showing the performance and benefits of running the rCUDA framework. The main advantage is that all the GPUs in the cluster are exposed as a big pool of resources for each node.

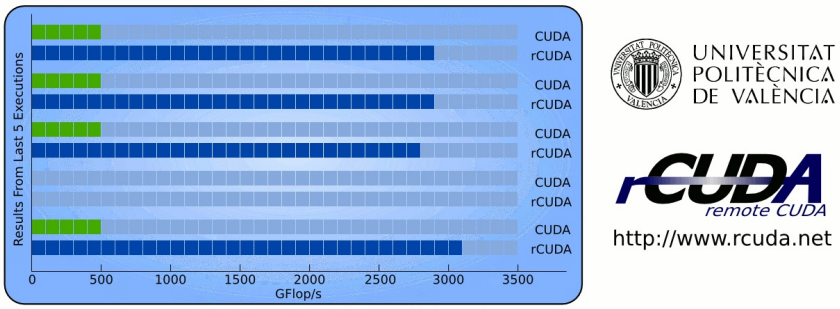


Figure 8. rCUDA performance running on 18 GPUs as compared to standard CUDA running on 1 GPU

The demo was eventuallly configured to run on IPoIB. For reasons that were not determined before the beginning of the conference the IB version of rCUDA was always freezing in an initialization stage. The presence of the Obsidian R400 router and BGFC and different OFED stacks were initially thought to be the problem. However even after the router was removed from the configuration and the OFED stacks synchronized the rCUDA was not able to work in native IB mode although other tests like ib_pingpong were succesful. rCUDA developers suggested that further tests are run after SC16 in order to determine why their framework is not able to function properly in a native long range IB setup.

1.3.4. Demonstrations with Poznań Supercomputing and Networking Centre (PSNC)

Vitrall (`http://apps.man.poznan.pl/trac/vitrall-test`) is a distributed web based visualization system. It is under development at the Applications Department of the Poznań Supercomputing and Networking Center.

A typical scenario of Vitrall usage is a real-time visualization of a complex 3D content using remote servers equipped with modern multi GPU solutions, like nVidia Tesla. Following frames are compressed as JPEG pictures and sent over HTTP protocol to clients. Still, they may be displayed on an attached screen or projector. Users may watch the same content from different points of view simultaneously. Information about users' input is sent back to the Vitrall Visualization Server using WebSocket protocol – part of HTML5 specification.

Rendering process was distributed among two locations: Poznań and Singapore - every second frame will be rendered in Singapore, and every other frame in Poznań and then accessed by client through Singapore. At the exhibition floor in SC a regular web browser was used to connect to the Vitrall instances and and visitors were able to interact with the presented 3D scene providing a smooth animation. Web browser uses WebSocket to connect with Vitrall server controller instance and after a new rendering session is established, client starts to send input messages. Controller instance interprets those messages and continuously applies changes to the authoritative state of presented 3D scene. That state is then incrementally replicated to both rendering instances - only those need access to a GPU device. One such instance runs locally with the controller instance (in Singapore), and the other runs in Poznań. Controller instance decides which frame to render where, sends rendering requests to rendering instances and notifies the client where following frames will be available. The client then requests those frame using HTTP in the way that frame rendered in Poznań are accessed through Singapore.

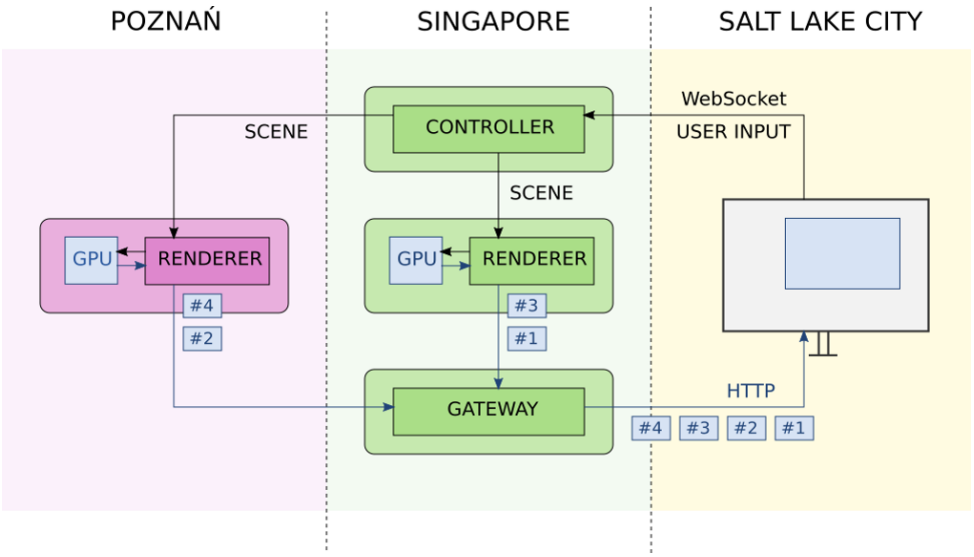


Figure 9. Vitrall distributed web based visualization system

Accomplishments and problems encountered:
The demo was succesfully run between PSNC and A*CRC without any major problems. Figure 10 shows a screenshot of the demo running in SC16. The performance was quite good as the interactivity with the scene was almost seameless. The only problem is that all servers involved in the demo requier quite powerfull GPUs.

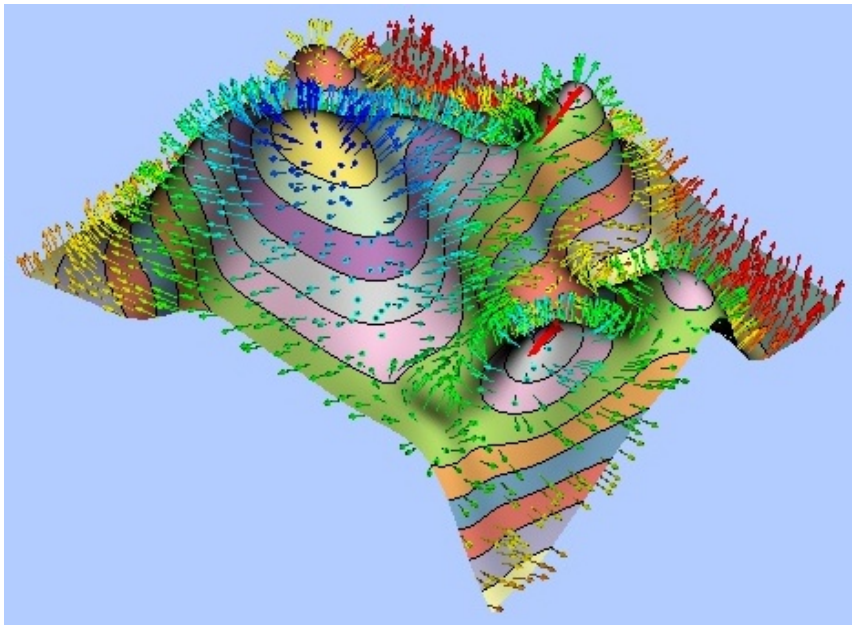


Figure 10. PSNC demonstration screenshot

1.3.5. Demonstrations with Interdisciplinary Centre for Mathematical and Computational Modelling (ICM)

The concept of demo was to show a basic proof-of-concept solution for globally remote visualization, where the simulation (or datasets), the visualization pipeline and the user are globally dispersed and connected only by a global network like InfiniCortex. A scientific example chosen for this demo was the numerical weather forecasting. The model itself (run by the Interdisciplinary Centre for Mathematical and Computational Modelling at the University of Warsaw (ICM)) is an iterative process of predicting an atmosphere state condition dynamics based on initial weather. In each step a significant number of data is created being a multivariate dataset on a three dimensional non-uniform grid over a modelled area. Observations of the evolution of a running simulation are possible by visualization of the consecutive iterations and provide the insight to the simulation. For the purpose of this a demo a dynamics of cloud coverage over central Europe was chosen. From the implementation perspective the demo consists of three layers: a simulation layer (or data layer), a visualization layer and end-user layer. The three layers were globally spatially disjoint and combined by a global interconnect. The simulation layer was physically located in Singapore (ACRC) and the running simulation was mimicked by incremental creation of new time step data files in the shared filesystem (each new file represents a simulation dump of a single iteration). The filesystem based on BeeGFS was remotely shared via InfiniCortex network and used by the visualization layer to access datasets. The visualization layer was physically located in Poland (ICM, Warsaw) and based on a dedicated visualization server running both the remote visualization middleware and the visualization software. VisNow (`http://visnow.icm.edu.pl`) was used as the visualization platform for implementation of the whole visualization pipeline. Visualization was created to show the orography of central Europe (static baseline layer) and the semi-transparent representation of cloud coverage was animated looping over the available iterations. A dedicated data access module in VisNow was monitoring the remote filesystem for presence of new time steps. Incremental dataset diffs were read in remotely via a shared filesystem. The remote visualization middleware was based on NICE Desktop

Cloud Visualization (DCV) platform, providing a remote desktop with dedicated data streaming for 3D OpenGL graphics and hardware compression. The end-user layer, being the interactive visualization, was physically located in the USA (SC16, Salt Lake City) and was running on a DCV thin client. On the DCV client-server path both InfiniCortex and Internet connections were tested.

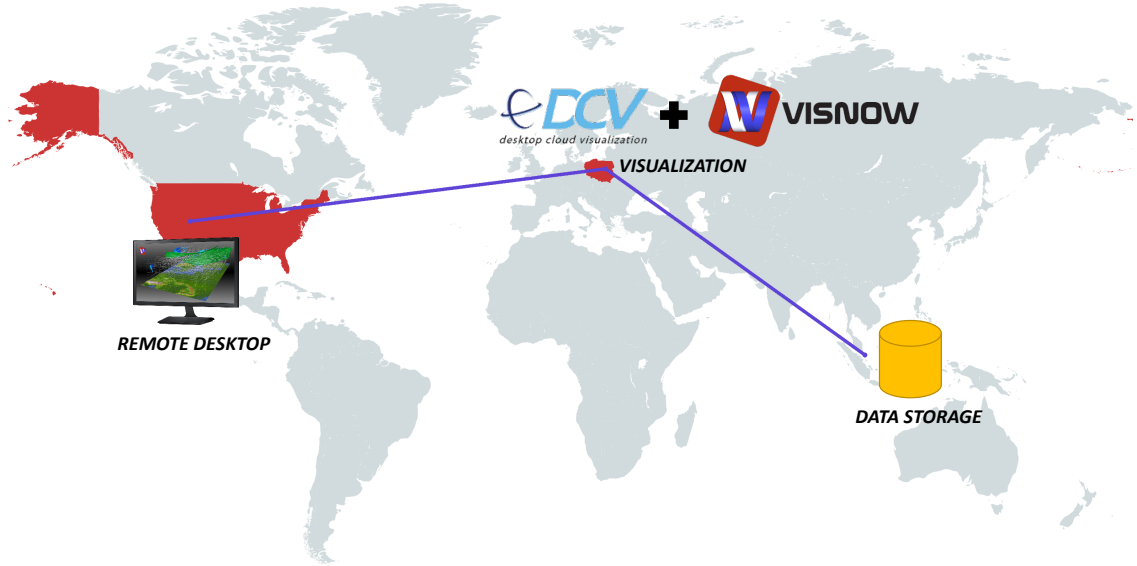


Figure 11. Remote visualization workflow

Accomplishments and problems encountered:
The proposed demo was successfully configured and run on a basic dataset of numerical weather forecast. As a proof-of-concept solution the demo showed the possible application of a globally connected supercomputer based on InfiniCortex network. At the same time novel knowledge was gathered on technical bottlenecks of the proposed visualization ecosystem and several concepts of improvement solutions were defined.

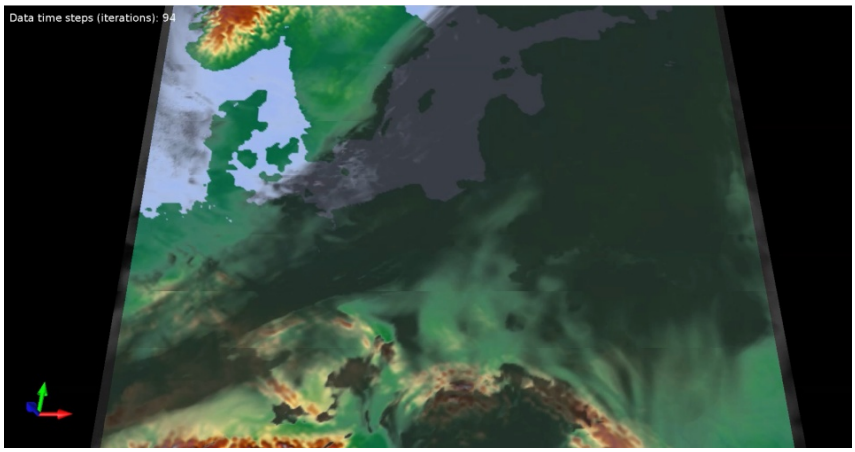


Figure 12. ICM demonstration screenshot

Some problems were encountered due to the fact that a BeeGFS parallel file system was being mounted over the InfiniCortex. This configuration was not tested and fine tuned before the demonstration leading to access time outs to the file system.

2. InfiniCortex Phase 2

Although over the last three years the InfiniCortex project ran as a proof-of-concept to demonstrate several unprecedented features, currently it is entering phase "InfiniCortex 2" where many of the features demonstrated start being integrated in production systems creating symbiotic collaborations and developing new projects [11].

At the beginning of the project the international connectivity was obtained through the goodwill of several international carriers. Since 2015 Singapore has its own permanent links with US and Europe dedicated to research. The National Supercomputing Centre of Singapore (NSCC) currently funds the enhancement of the connectivity and through SingAREN MoUs have been signed for co-funding of permanent international links (100 Gbps towards US with Internet2 and 10Gbps towards Europe with TEIN*CC). These links will open the world and benefit the entire research community in Singapore, thus creating a symbiotic relationship between several local entities. The link to US set to open new oppotunities such as the participation in the Global Research Platform, an international extension of the Pacific Research Platform [3]. Similar international connectivity are currently being negotiated with Australia and Japan.

In the next years, NSCC and A*CRC are set to work on implementing a nation-wide InfiniBand fabric to interconnect several academic and industrial sites in Singapore which will provide high throughput, low latency direct connection to the supercomputing facilities in Fusionopolis. The first steps have already been made with the launch of the NSCC who has its main stakeholders campuses (NUS, NTU, GIS) connected through InfiniBand directly to the main supercomputer. This infrastructure provides researchers in remote campuses an unparalleled fingertip access to HPC resources. This initiative to allow a wide access to the HPC resources will continue in the future under NRF funding for National Research Infrastructure.

Genome Institute of Singapore (GIS) has the largest sequencing facility in South-East Asia at their facilities at the Genome Building, Biopolis. GIS relies on in-house storage as well as storage in Matrix Building Biopolis (100m away) and on storage and high performance compute facilities 2km away at the A*STAR Computational Resource Centre, Fusionopolis. Going forward, GIS will be processing up to thousands of exomes on a regular basis, processing capacity needs to be ramped up to cope with this demand of several Terabytes of data per day emerging from their sequencing labs. This means that in-house generated sequence data must be safely stored for data regulation compliance reasons, as well as transferred and stored at remote location pending computational processes such as the quality control step, read mapping step (high memory), variant call steps (embarrassingly parallel) and annotation steps involving different types of software with different hardware requirements. To avoid a data-bottleneck, we have constructed on top of standard TCP/IP network of A*STAR's next generation ExaNet a 500Gbps Infinera CloudExpress 1 Ethernet link plus a Mellanox MetroX and Obsidian Longbow InfiniBand interconnections between Biopolis and Fusionopolis. The 500 Gbps link runs over a dedicated dark fibre and is the fastest point-to-point link in Asia, and the fastest known link in the world dedicated for Next Generation Sequencing (NGS) analytics. By 2017 the whole bandwidth capacity between Biopolis and Fusionopolis will exceed 1.2 Tbps. In 2016, a 1 Terabyte RAM node was installed in GIS Biopolis linked by InfiniBand to the new NSCC supercomputer

with 10 PByte storage (up to 500Gbytes/sec I/O using DDN's state-of-the-art Infinite Memory Engine, accessing a dedicated genome analytic queue on *ASPIRE 1*, the 1PFLOPS NSCC supercomputer at Fusionopolis. This will become one the world fastest and biggest distributed genome processing system and will ensure that genome analytics can be scaled up to thousands of genomes to be processed routinely per month for biomedical research which will be a current practice in the framework of projects like Genome Asia 100k.

Conclusions

InfiniCortex project started by A*CRC under leadership of Dr Marek Michalewicz in 2014 has gained a lot of interes both in Singapore and abroad. The A*CRC group which was responsible for the InfiniCortex project was recognized through several awards during the past few years, the most prestigious one being the *Innovative Project Gold Award* from the Ministry of Trade and Industry of Singapore in 2015 [1]. InfiniBand connectivity is still regarded as a high-end HPC oriented interconnect however features such as high-bandwidth and security which are demonstrated advantages over the classic TCP/IP are now recommending it for production environments outside the walls of a datacentre.

InfiniCortex project could serve as a very useful prototypical infrastructure for a number of Big Scientific Data projects currently being developed: i) distribution of data to the international partners from the Square Kilometre Array [4], ii) streaming of large facilities experimental data through the Pacific Research Platform collaboration of DoE, ESnet and other partners in the US and elsewhere [3], iii) the Superficilities vision expressed by DoE [6], and iv) new architecture for CERN LHC data processing pipeline focussing on more powerful processing facilities connected by higher throughput connectivity [13] .

Data connectivity between key HPC centres and countries has been defined as one of the priority areas of newly established EuroHPC programme. *"HPC is developing to cope with the constant increase in data volumes and flows. A recent report projects that annual global IP traffic will reach 2.3 zettabytes by 2020 – or 504 billion DVDs per year."* [2]

The authors are firmly convinced that *InfiniCortex I* project provides very well defined and tested path to realising some of the goals of EuroHPC connectivity agenda.

Acknowledgements

Authors would like to acknowledge all the partners around the world that supported and worked hard to make the InfiniCortex project a success.

USA:

ORNL: Scott Klasky, Jong Youl Choi, Matthew Wolf, Dave Pugmire, Manish Parashar

Fermilab: Wenji Wu, Jason Wang, Liang Zhang

Stony Brook University: Robert Harrison, Firat Coskun, Behzad Barzideh, Michael White, Yuefan Deng, Tahsin Kurc

George Washington University

Adam Kai Leung Wong, Jaroslav Flidr, Donald DuRousseau, Andrew Gallo

France: University of Reims Champagne-Ardenne

Arnaud Renard, Michaël Krajecki

Spain: Universitat Politècnica de València

Carlos Reaño, Federico Silla

Poland:
Poznań Supercomputing and Network Centre (PSNC): Artur Binczewski, Tomasz Szewczyk, Tomasz Piontek, Piotr Śniegowski
Interdisciplinary Centre for Mathematical and Computational Modelling (ICM), University of Warsaw: Marek Michalewicz, Bartosz Borucki, Konrad Bierzuński, Jaroslaw Skomial
Singapore:
SingAREN: John Kan, Francis Lee, Lawrence Wong
Commercial Partners:
Obsidian: David Southwell, Jason Gunthorpe, Bill Halina
DDN: Atul Vidwansa, Susan Presley
Huawei: Chenxingying Nick, Robin Shi Bin

References

1. A*CRC team received five awards in 2015:. Singapore Ministry of Trade and Industry Awards 2015: Innovative Project Gold Award 2015; A*STAR Awards 2015: STAR Innovation Award 2015; FutureGov Singapore Award: Technology Leadership Award; CIO Asia 100: Honouree 2015; Singapore Public Service Most Innovative Project: Merit Award 2015.

2. Eurohpc. https://ec.europa.eu/digital-single-market/en/news/eu-ministers-commit-digitising-europe-high-performance-computing-power.

3. Pacific research platform. http://prp.ucsd.edu/. Last accessed: May 10, 2017.

4. Square kilometer array. http://skatelescope.org/signal-processing/. Last accessed: May 10, 2017.

5. Obsidian introduces the Bowman Global Fabric Controller. http://www.obsidian research.com/archives/all/2015/Bowman-Global-Fabric-Controller.html, November 2015. Last accessed: May 10, 2017.

6. Kate Antypas. Superfacility: How new workflows in the DOE Office of Science are influencing storage system requirements. http://storageconference.us/2016/Slides/KatieAntypas.pdf, May 2016.

7. Kenneth Hon Kim Ban, Jakub Chrzeszczyk, Andrew Howard, Dongyang Li, and Tin Wee Tan. Infinicloud: Leveraging the global infinicortex fabric and openstack cloud for borderless high performance computing of genomic data. *Supercomputing Frontiers and Innovations*, 2(3):14–27, 2015.

8. Jakub Chrzeszczyk, Andrew Howard, Andrzej Chrzeszczyk, Ben Swift, Peter Davis, Jonathan Low, Tin Wee Tan, and Kenneth Ban. Infinicloud 2.0: distributing high performance computing across continents. *Supercomputing Frontiers and Innovations*, 3(2):54–71, 2016.

9. Jonathan Low, Jakub Chrzeszczyk, Andrew Howard, and Andrzej Chrzeszczyk. Performance assessment of infiniband hpc cloud instances on intel haswell and intel sandy bridge architectures. *Supercomputing Frontiers and Innovations*, 2(3):28–40, 2015.

10. Marek Michalewicz, David Southwell, Tin Wee Tan, Yves Poppe, Scott Klasky, Yuefan Deng, Matthew Wolf, Manish Parashar, Tahsin Kurc, C.S. Choong-Seock Chang, Satoshi Matsuoka, Shin'ichi Muira, Jakub Chrzęszczyk, and Andrew Howard. InfiniCortex: concurrent supercomputing across the globe utilising transcontinental InfiniBand and Galaxy of Supercomputers. *Supercomputing 2014: The International Conference for High Performance Computing, Networking, Storage and Analysis, At New Orleans, LA, USA*, November 2014.

11. Marek T Michalewicz, Tan Geok Lian, Lim Seng, Jonathan Low, David Southwell, Jason Gunthorpe, Gabriel Noaje, Dominic Chien, Yves Poppe, Jakub Chrzęszczyk, et al. Infinicortex: present and future invited paper. In *Proceedings of the ACM International Conference on Computing Frontiers*, pages 267–273. ACM, 2016.

12. Ł Orłowski, Yuefan Deng, and M Michalewicz. Galaxies of supercomputers and their underlying interconnect topologies hierarchies. In *International Supercomputer Conference, Leipzig, Germany*, 2014.

13. Gianfranco Sciacca. Big Data Science Accessing High-End HPC. https://www.digital infrastructures.eu/sites/default/files/LHConCRAY-DI4R2016-v2.pdf, October 2016.