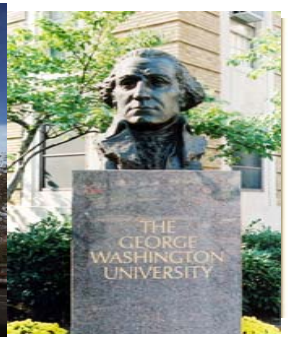


Hierarchical Locality and Parallel Programming in the Extreme Scale Era

Tarek El-Ghazawi

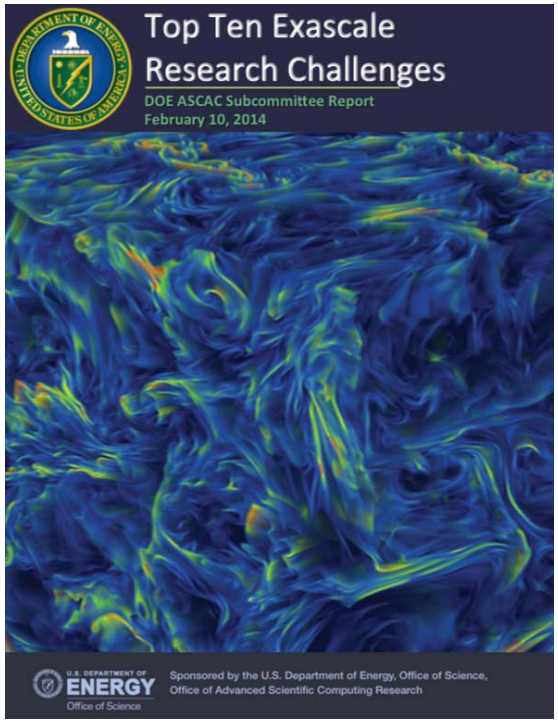
The George Washington University

University of Southern California
September 29, 2016

Overview

- **Fundamental Challenges for Extreme Computing**
- Locality and Hierarchical Locality
- Programming Models
- Hardware Support for Productive Locality Exploitation- Address Remapping
- Hierarchical Locality Exploitation
- Concluding Remarks

Top Ten Challenges for Exascale: Areas where Research and advances are needed!



DoE ASCAC Subcommittee Report Feb 2014

1. Energy Efficiency ✓
2. Interconnect Technology ✓
3. Memory Technology ✓
4. Scalable System Software ✓
5. Programming Systems ✓
6. Data Management ✓
7. Exascale Algorithms
8. Algorithms for Discovery, Design & Decision
9. Resilience and Correctness
10. Scientific Productivity ✓

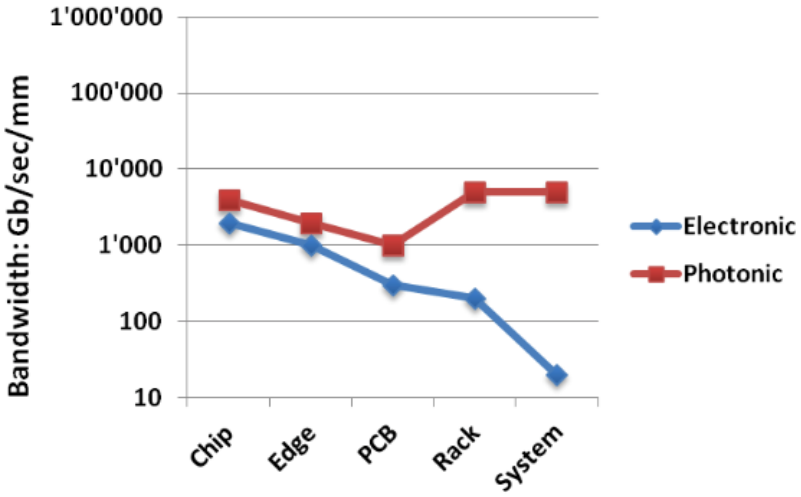
HPCL The George Washington University High Performance Computing Lab

✓ **Data movement and/or programming related**

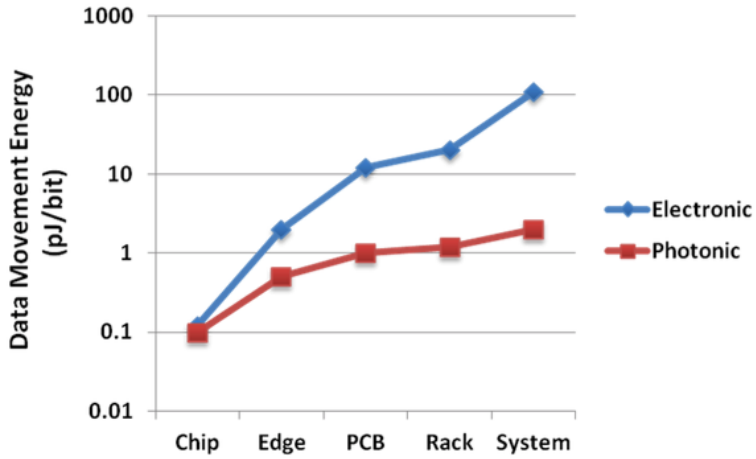
THE GEORGE WASHINGTON UNIVERSITY WASHINGTON, DC

Technological Challenges: Combined Bandwidth and Energy Challenges for Exascale

Bandwidth density vs. system distance



Energy vs. system distance

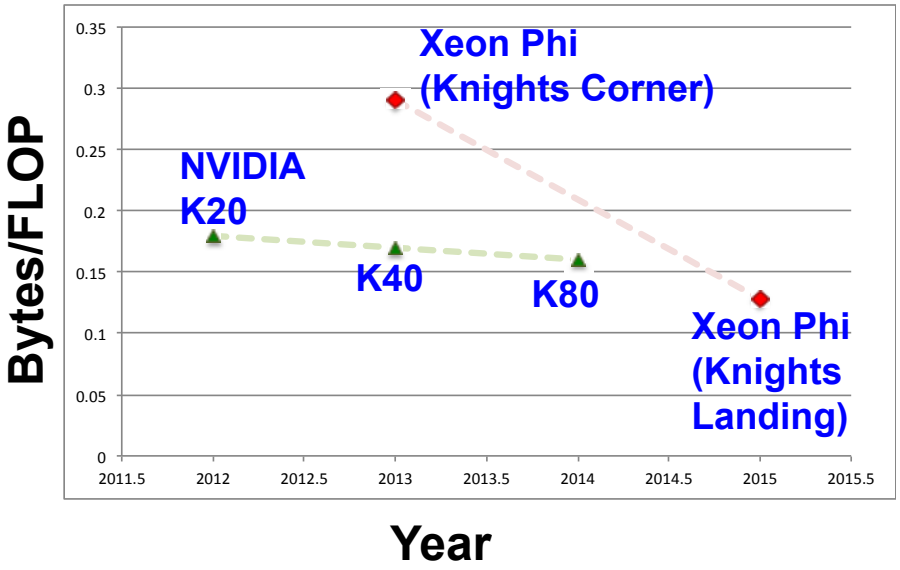


[Source: ASCAC 14]

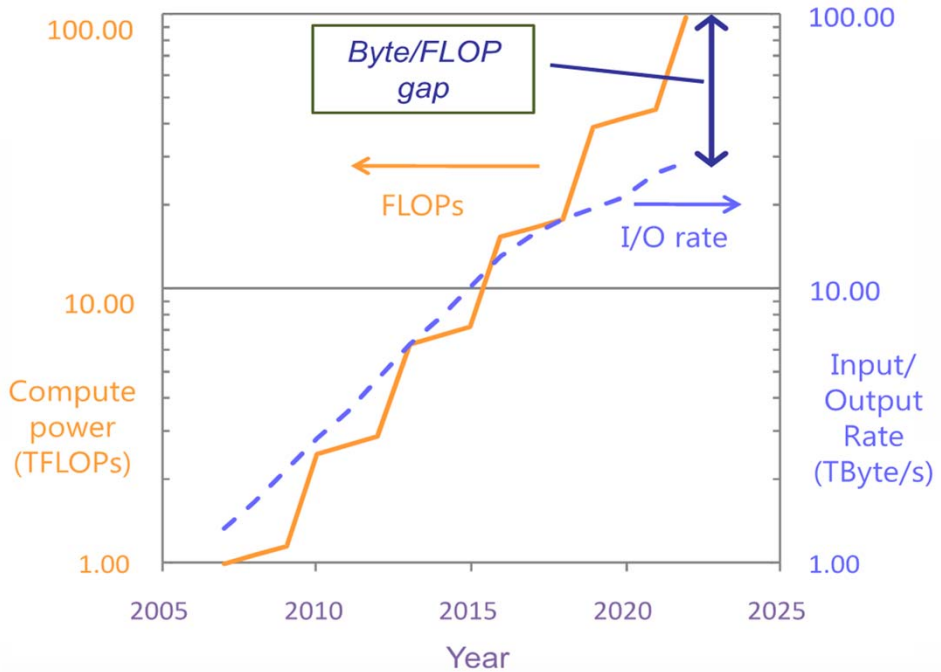
- Locality and data movement matter a lot, cost (energy and time) rapidly increases with distance
- Locality and data movement are critical even at short distance, more so at far distances

Technological Challenges : (2) Bandwidth

Growing manycore bandwidth requirements



Widening gap between available I/O and compute capability



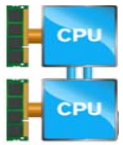
Ref: Miller, D. A, *Proceedings of the IEEE*, 2009.

- Interconnect is not keeping up with the growth in compute capability
  - Many apps require 1 Byte/FLOP off-chip, not possible in 10 TFLOPs chips and beyond
    Intel Knights Landing: 500 GB/s => 1/6 Byte/FLOP
  - Huge bandwidth density (GB/s/μm) needed on-chip due to large #cores in small area

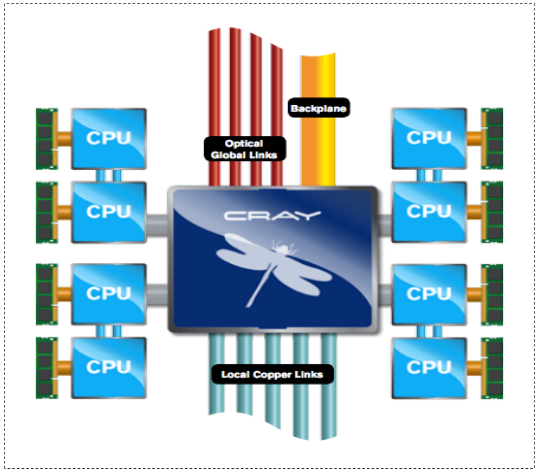
Overview

- Fundamental Challenges for Extreme Computing
- **Locality and Hierarchical Locality**
- Programming Models
- Hardware Support for Productive Locality Exploitation- Address Remapping
- Hierarchical Locality Exploitation
- Concluding Remarks

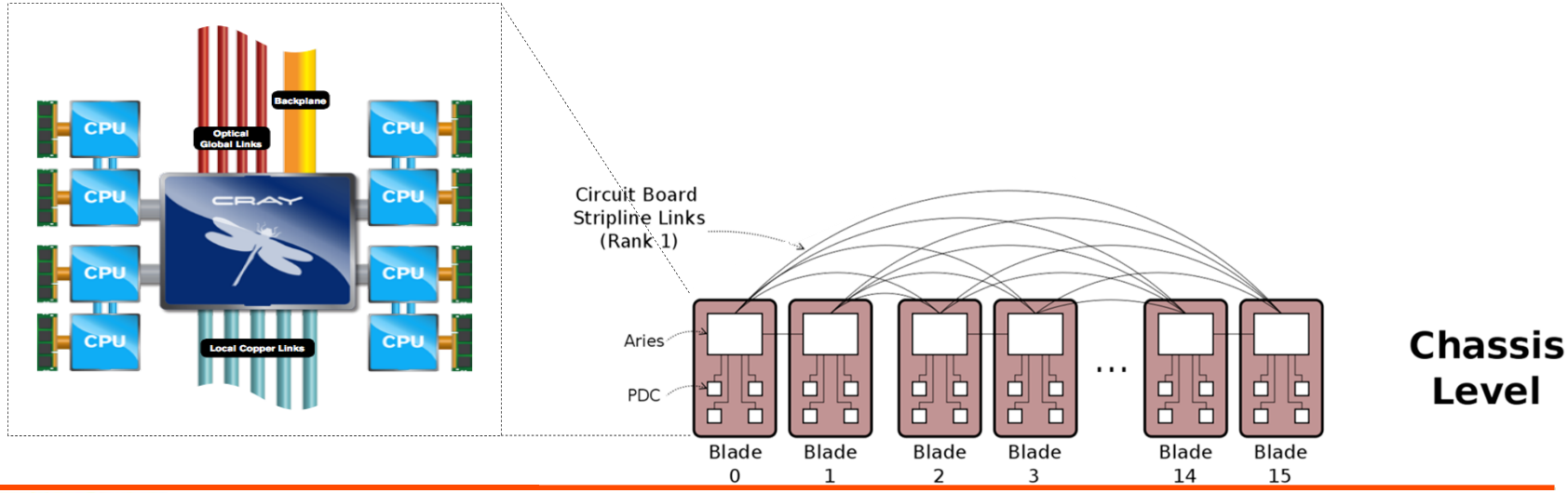
Architectural Challenges: Architectures are becoming Deeply Hierarchical in Extreme Scale – Chips and Systems



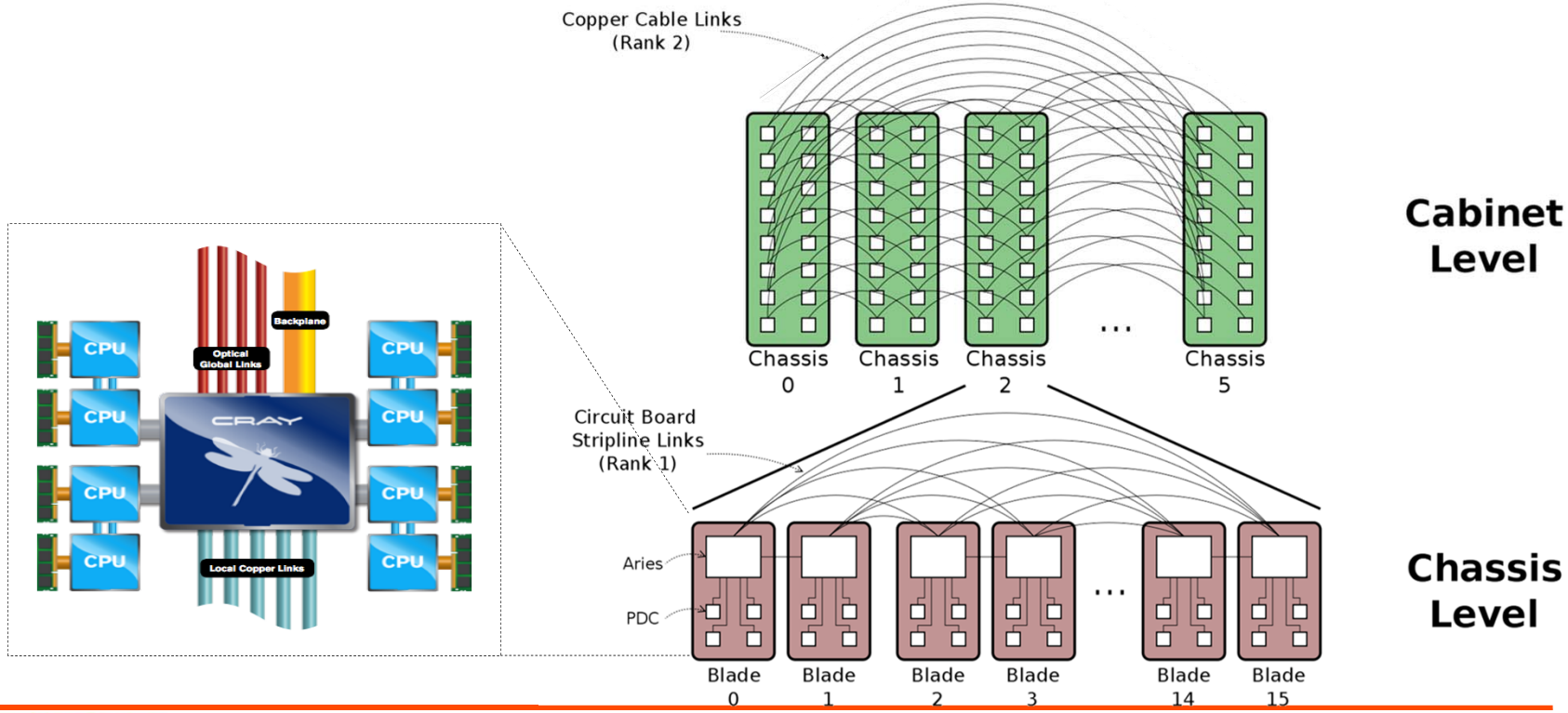
Architectural Challenges: Architectures are becoming Deeply Hierarchical in Extreme Scale – Chips and Systems



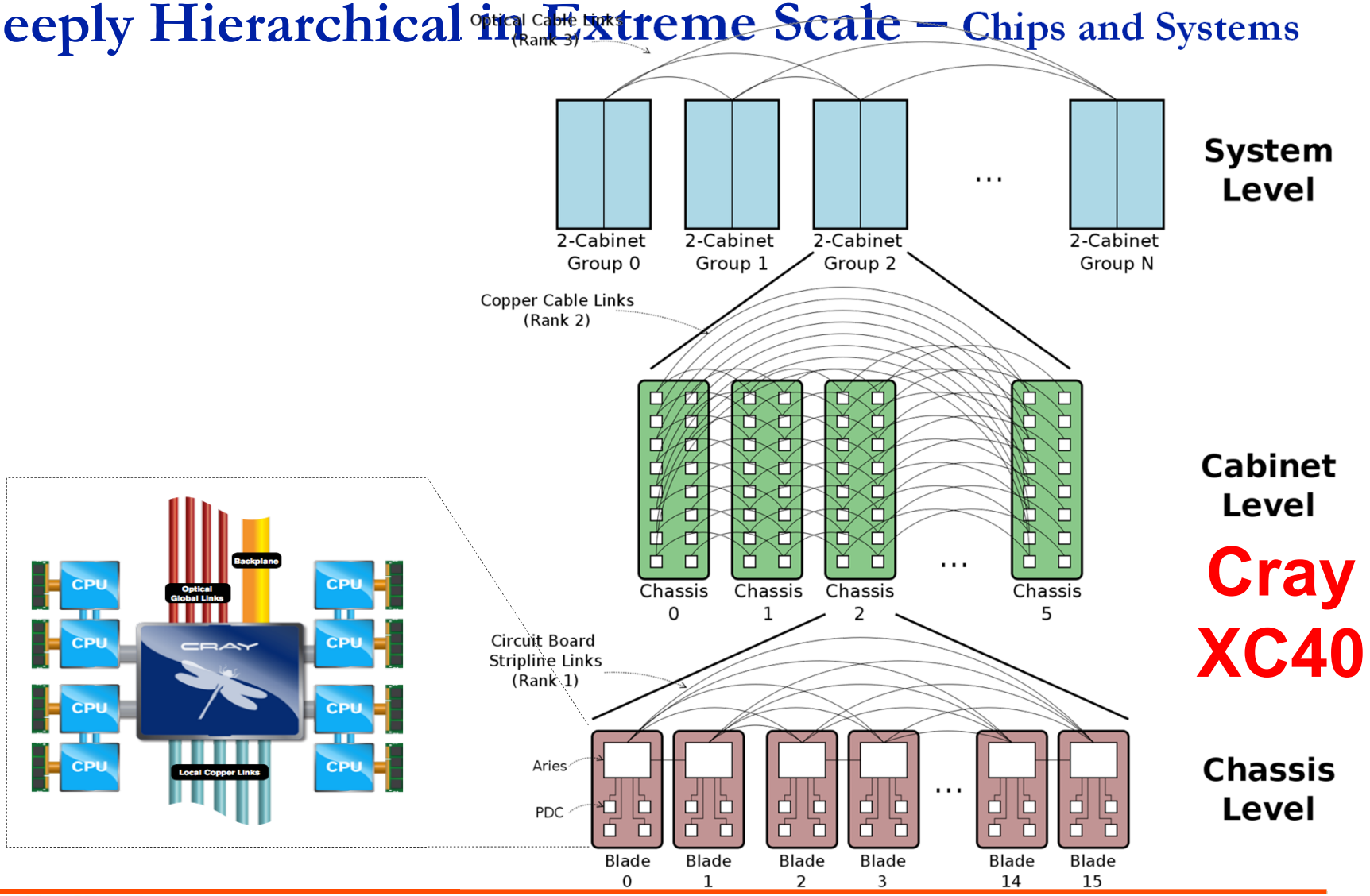
Architectural Challenges: Architectures are becoming Deeply Hierarchical in Extreme Scale – Chips and Systems



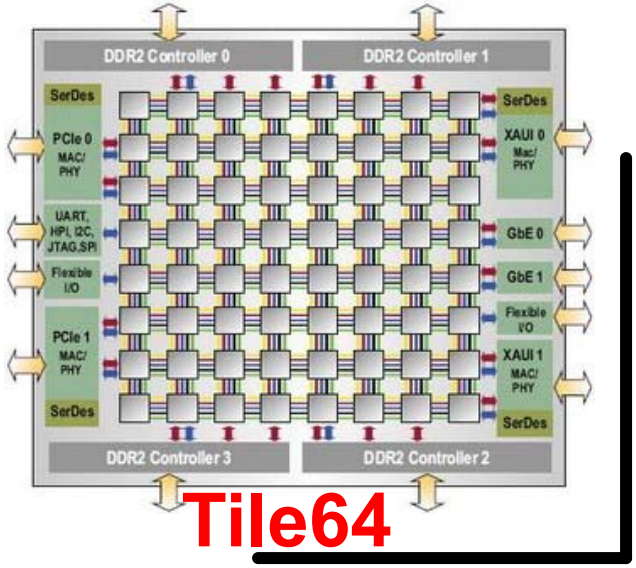
Architectural Challenges: Architectures are becoming Deeply Hierarchical in Extreme Scale – Chips and Systems



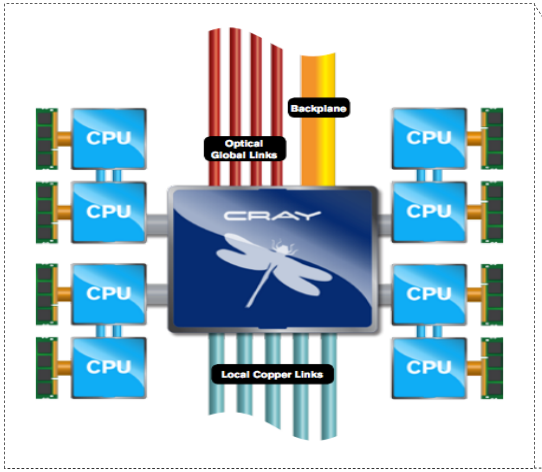
Architectural Challenges: Architectures are becoming Deeply Hierarchical in Extreme Scale – Chips and Systems

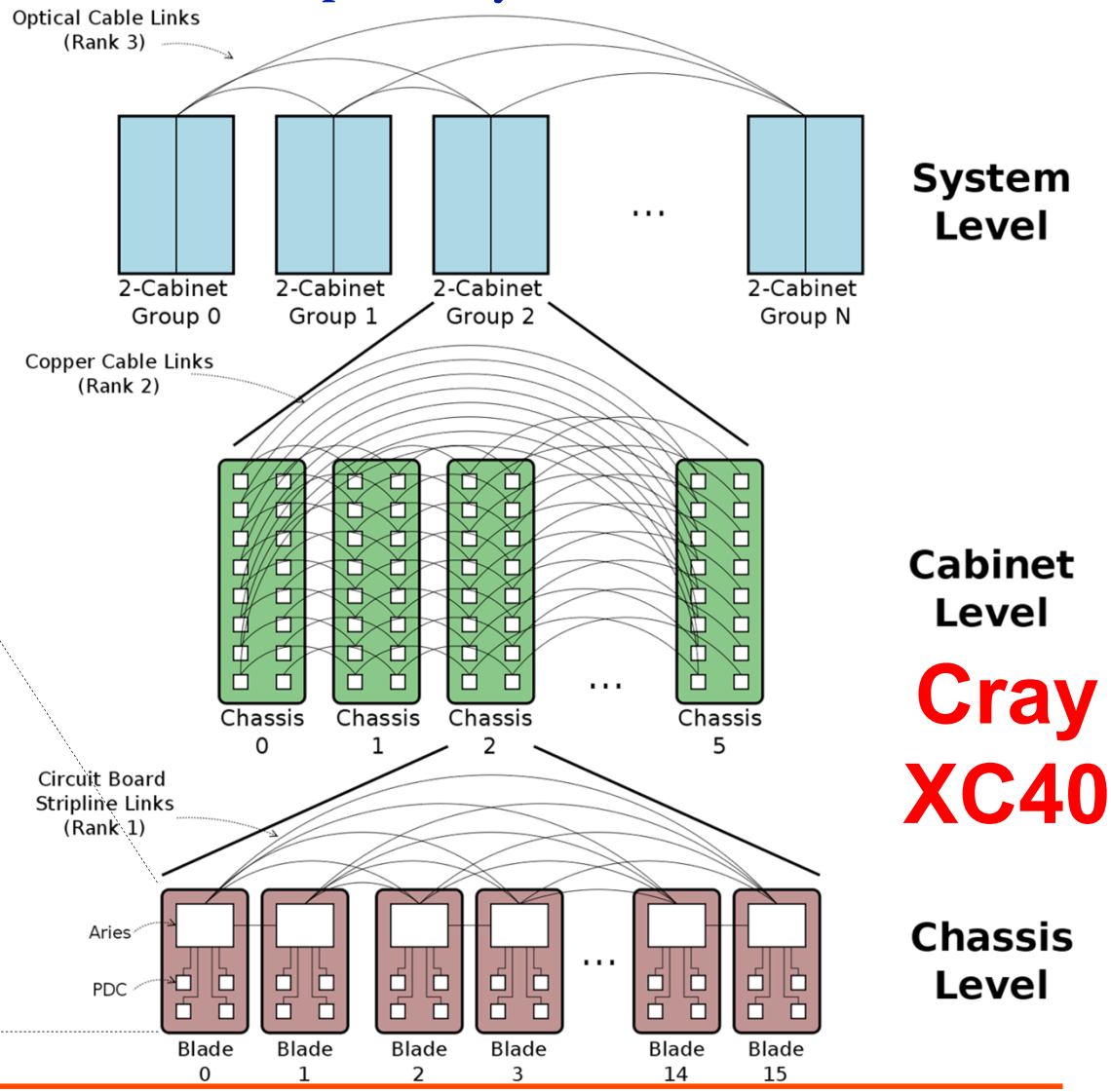


Architectural Challenges: Architectures are becoming Deeply Hierarchical in Extreme Scale – Chips and Systems



Tile64





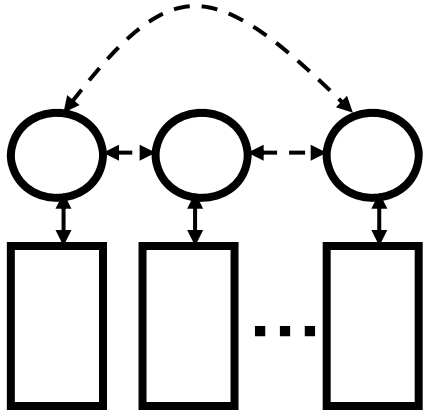
Overview

- Fundamental Challenges for Extreme Computing
- Locality and Hierarchical Locality
- **Programming Models**
- Hardware Support for Productive Locality Exploitation- Address Remapping
- Hierarchical Locality Exploitation
- Concluding Remarks

Where are Programming Models from That?

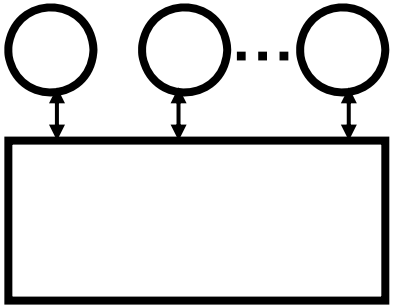
- What is a programming model?
  - An abstract virtual machine
  - A view of data and execution
  - The logical interface between architecture and applications
- Why Programming Models?
  - Decouple applications and architectures
    - Write applications that run effectively across architectures
    - Design new architectures that can effectively support legacy applications
- Programming Model Design Considerations
  - Expose modern architectural features to exploit machine power and improve performance
  - Maintain Ease of Use
  - Two previous points mean increase productivity!

Current Programming Models and Locality Awareness



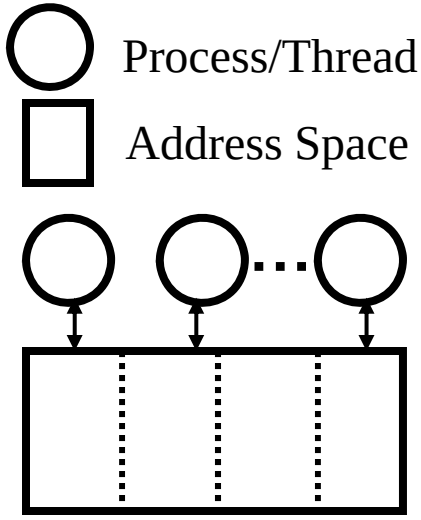
Message Passing

✔ Locality Awareness
-Two-Sided Communication
-Example MPI



Shared Memory

✘ Locality Awareness
-One-Sided Communication
-Example OpenMP



Partitioned Global Address Space

✔ Locality Awareness
-One-Sided Communication
-Examples UPC and Chapel

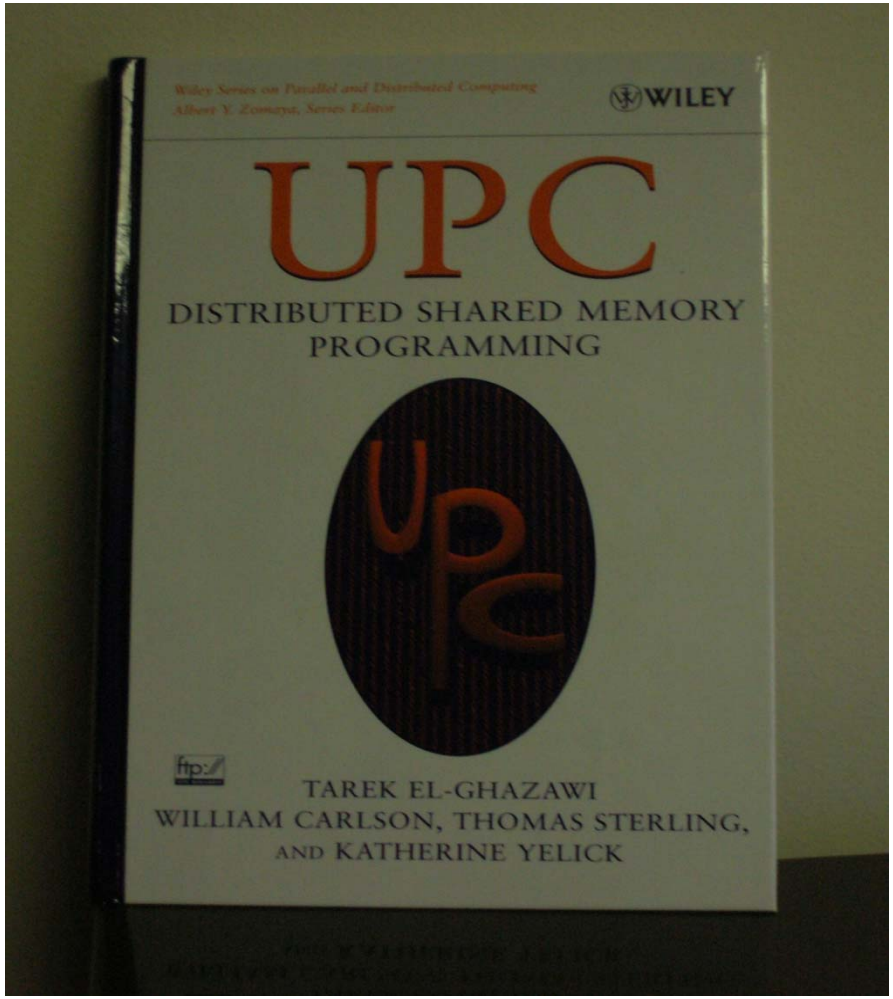
PGAS Languages Include UPC, Chapel and X10

UPC Language Specifications
V1.0

Tarek A. El-Ghazawi
George Washington Universtiy
tarek@gwu.edu

William W. Carlson Jesse M Draper
IDA Center for Computing Sciences
wwc@super.org jdraper@super.org

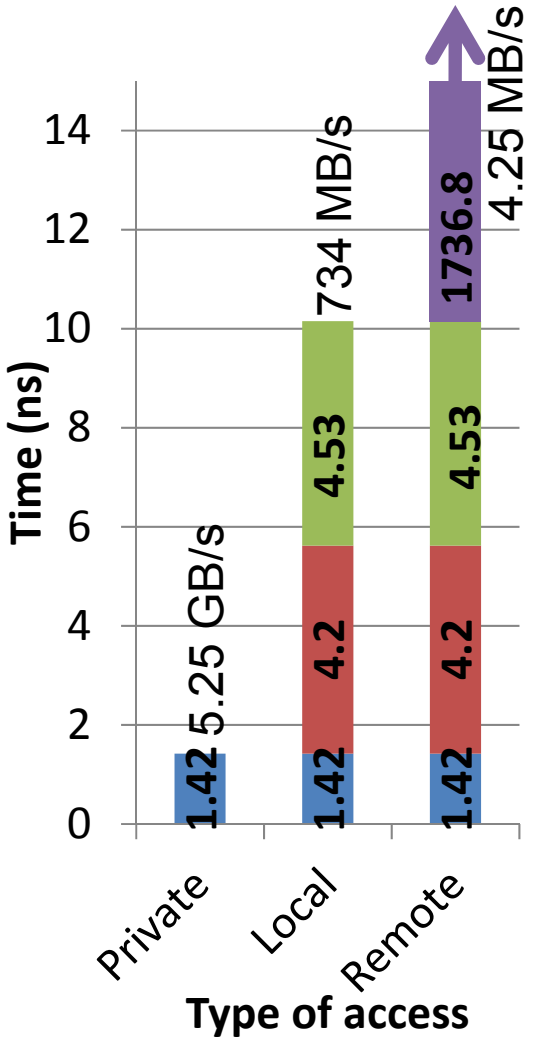
February 2001

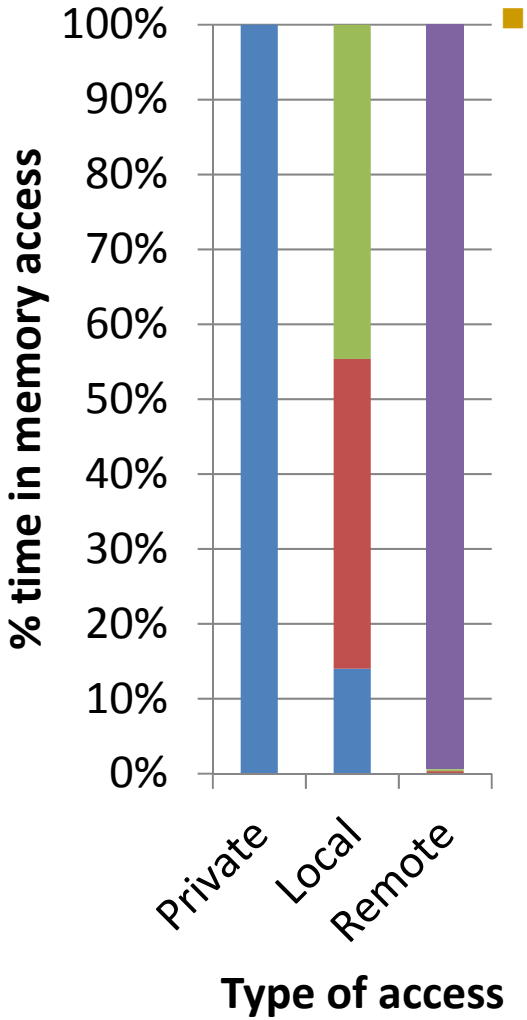


Overview

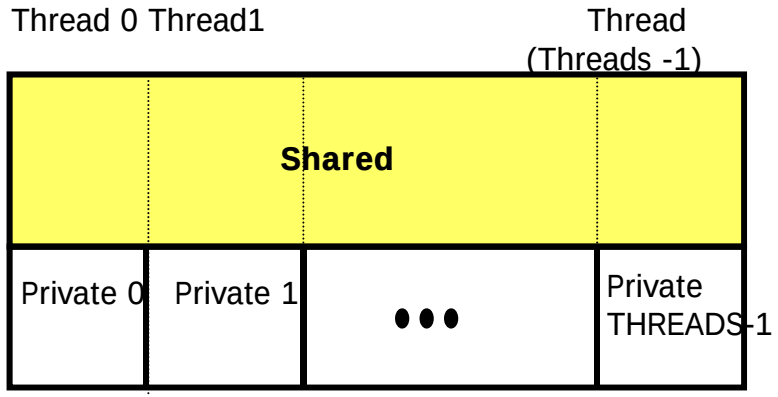
- Fundamental Challenges for Extreme Computing
- Locality and Hierarchical Locality
- Programming Models
- **Hardware Support for Productive Locality Exploitation- Address Remapping**
- Hierarchical Locality Exploitation
- Concluding Remarks

Memory Accesses in UPC- Shared Address Translation Overheads



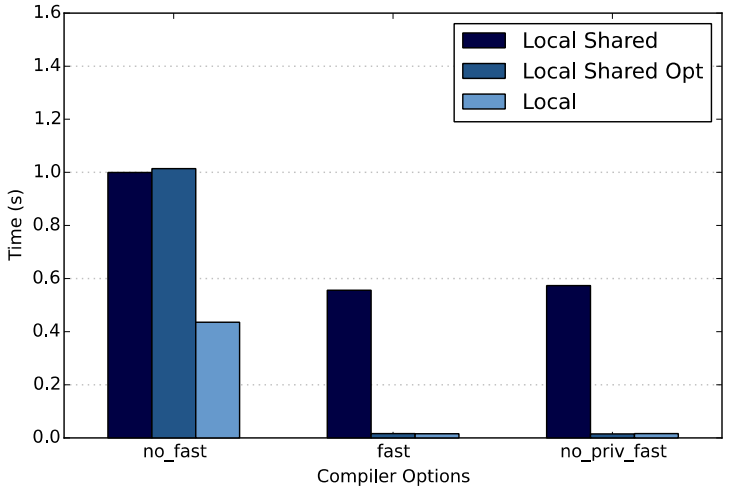


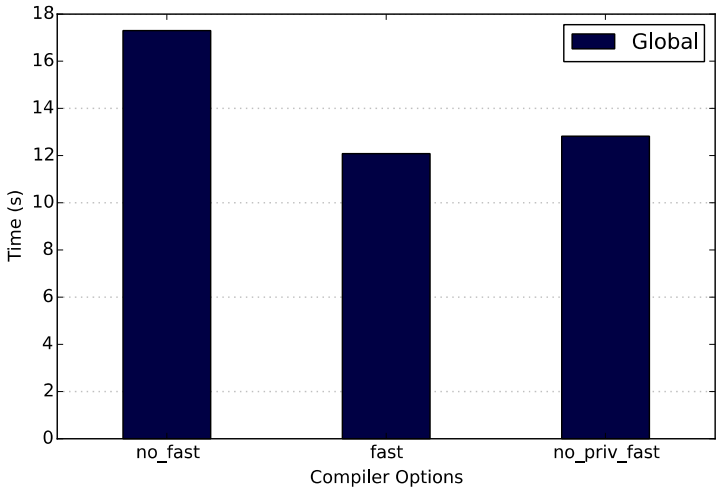
- **Measurement of the address space overheads**
  - **Set of micro-benchmarks measuring the different aspects separately:**
    - Network Time
    - Address Translation
    - Address Incrementation
    - Memory Access



Memory Access Costs in Chapel

- Tested shared address access costs in Chapel:
  - Used Chapel Syntax to test
    - Local part of a distributed object, un-optimized- Accessing local data without saying local
    - Local Optimized – local part hand-optimized by saying “local”
    - Local and Non-Distributed
- Compiler optimization -> 2x faster
- Both compiler and hand optimization -> 70x faster
- Compiler optimization affects remote accesses as well
- Both UPC and Chapel require “unproductive!” hand tuning to improve local shared accesses





Fast Address Translation for PGAS

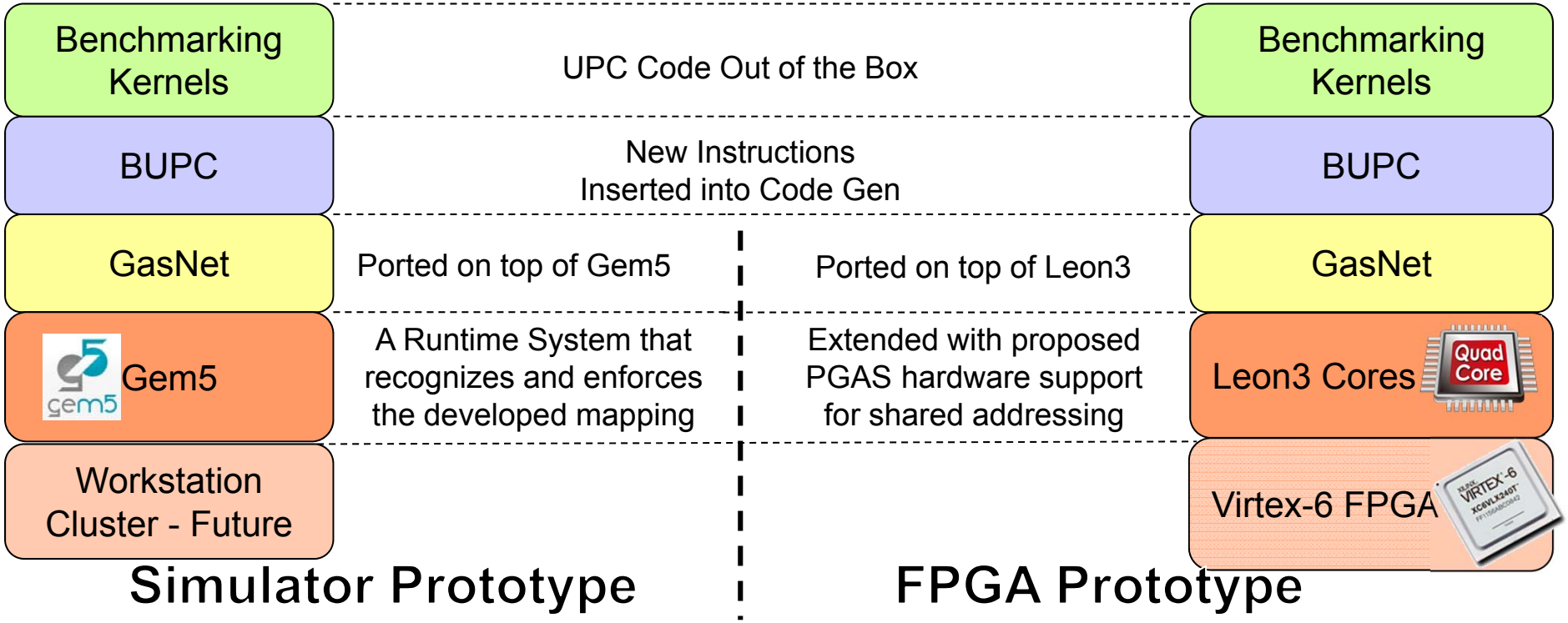
- Software solutions
  - Hand tweaking – Non-productive
  - Compiler optimizations - reduced arithmetic for some straightforward cases
  - Look up tables, full and reduced- Take memory! ICPP05
  - TLB's ....
- Hardware solutions
  - Create hardware that understands how to traverse the PGAS memory model and support basic costly needs
  - Avail it through instructions and leverage them by the compiler
- Some work for UPC, little for Chapel

Hardware Support for PGAS

- **Example Operations for Support in Hardware**
  - **Shared address incrementing**
  - **Load/store to/from a PGAS shared address**
    - Address translation support: convert a shared address to a system virtual address used to perform the access
  - **Locality tests for remote data**
    - Can be used to tell whether to call the network subroutines, by e.g. testing the affinity field in a work sharing construct
- **Availed as ISA extension**
- **New instructions used directly by compiler**
- **Current hardware support and instructions only support address mapping**
- **Future support for remote data accesses and various types of synchronizations are of interest**

Hardware/Software Co-Design Platform in a Nutshell

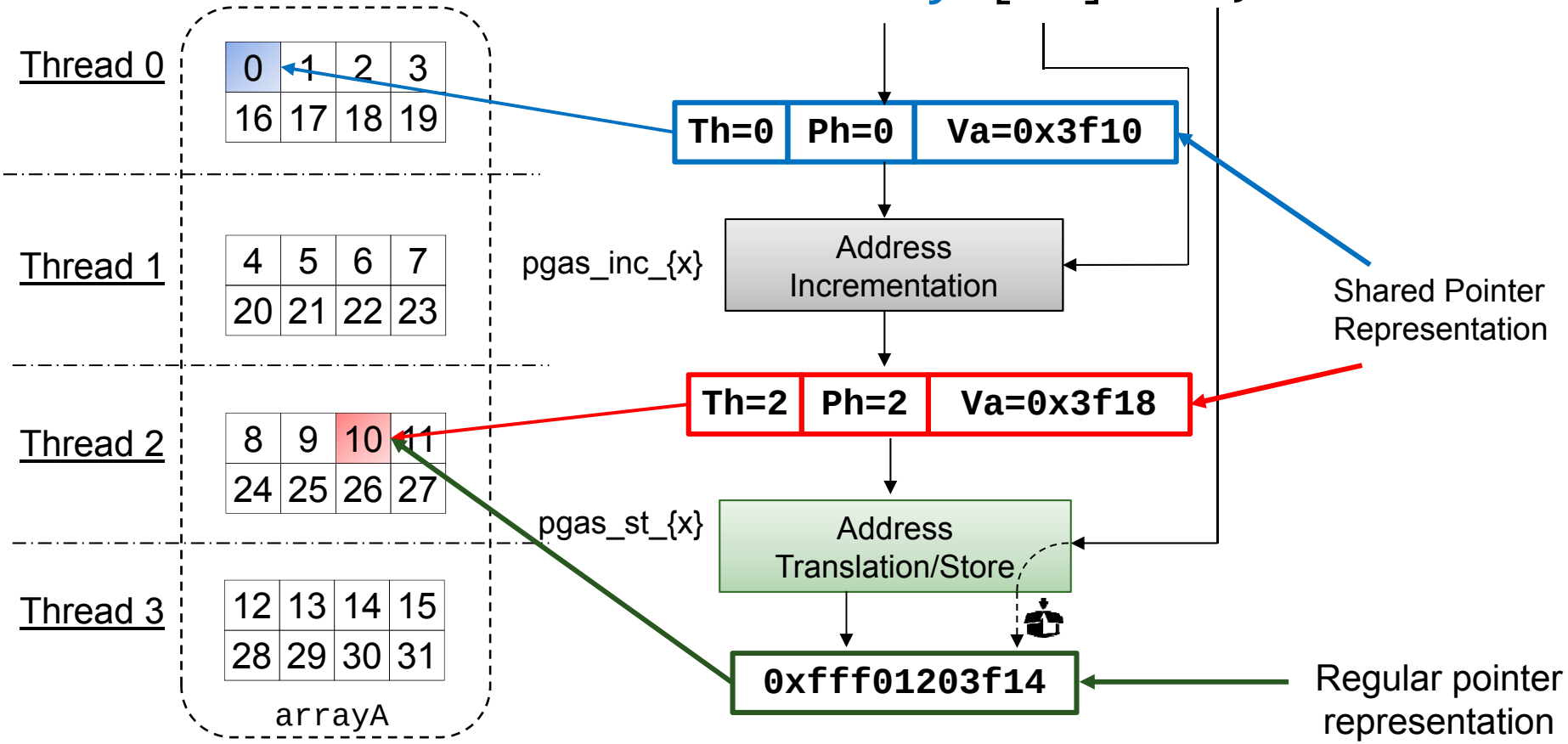
- First prototype in FPGAs, supports small core count and apps
- Second is primarily software, supports bigger core counts and codes



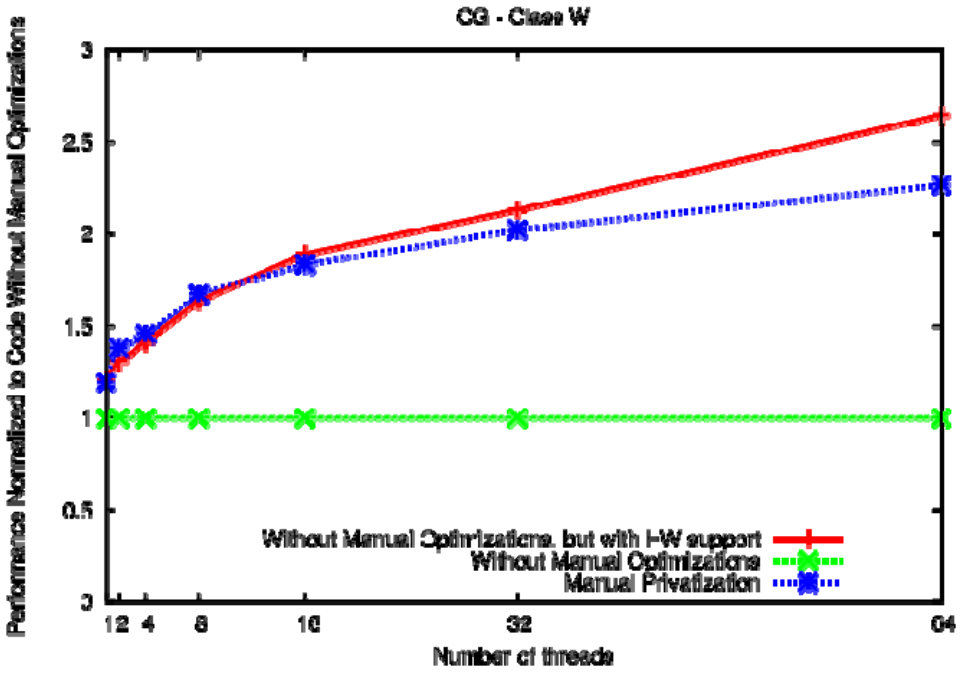
PGAS Hardware Support Overview

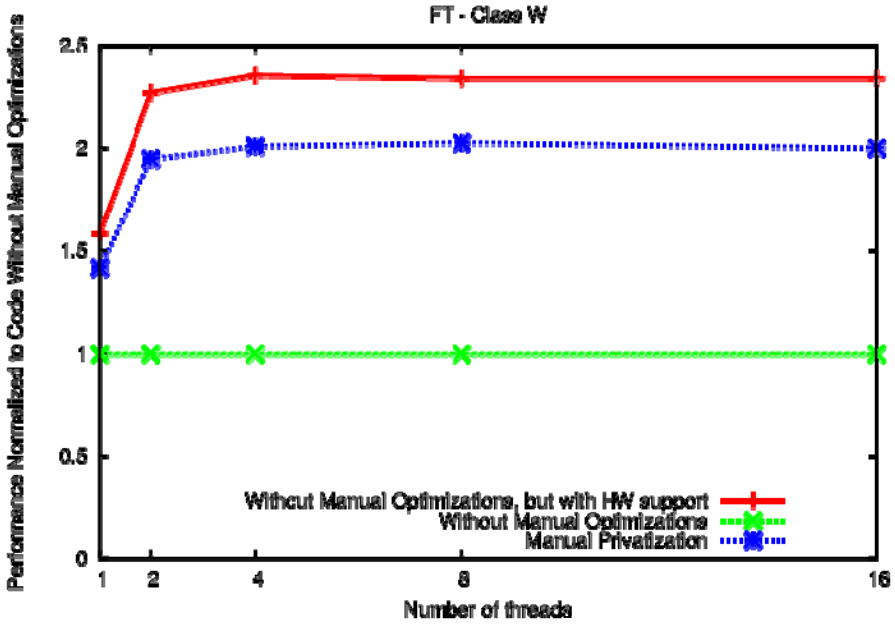
- `shared [4] int arrayA[32];`

```
arrayA[10] = 5;
```



Early Results- NPB Kernels with HW Support Gem5 Alpha 21264





Overview

- Fundamental Challenges for Extreme Computing
- Locality and Hierarchical Locality
- Programming Models
- Hardware Support for Productive Locality Exploitation- Address Remapping
- **Hierarchical Locality Exploitation**
- Concluding Remarks

Possible Solutions for Hierarchical Locality Exploitation

- Rewrite your code with low-level tricks to target the underlying hierarchical architecture?
  - Great performance, but not productive & non-portable
- Extend programming models with hierarchical syntax and semantics and ask programmers to worry about all of those hardware details? (make them hierarchical-locality-aware!)
  - Portable but not productive

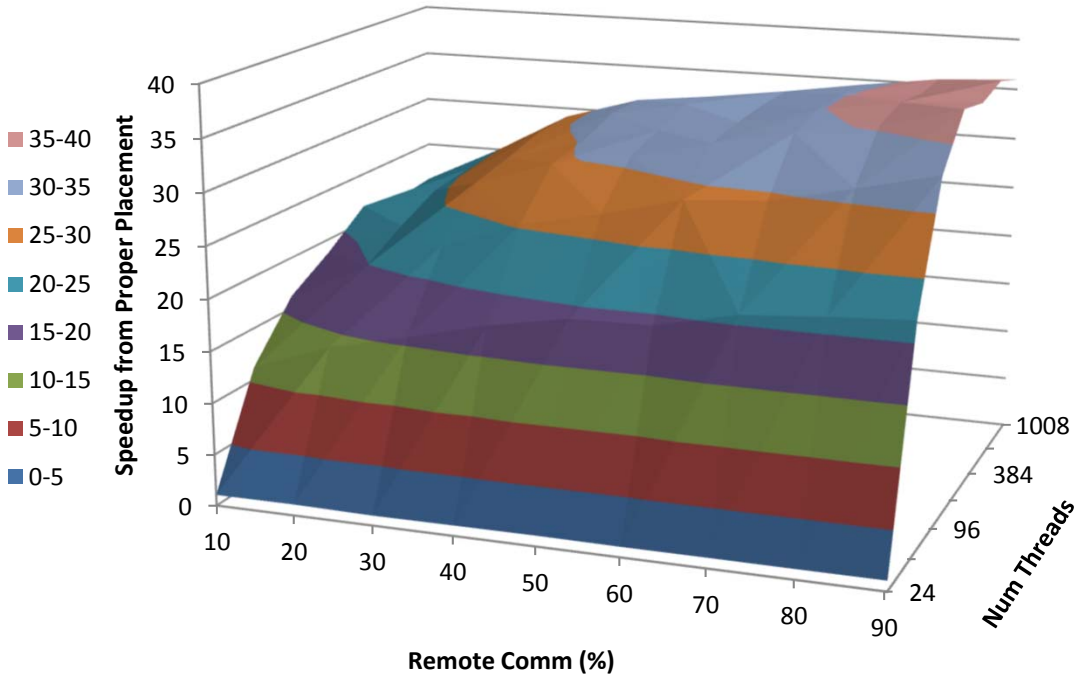
Productive Division of Responsibilities: The Programmer and the System

- Programmer
  - Use a locality-aware programming paradigm such as MPI or a PGAS language
  - Let programmer worry about the first-order locality, thread-data affinity
- System
  - Understand your system hierarchy, costs associated with data movements across levels
  - Understand the program characteristics
  - Derive locality exploitation on level-by-level basis via Hierarchical Thread Grouping/partitioning

Motivations and Early Investigations

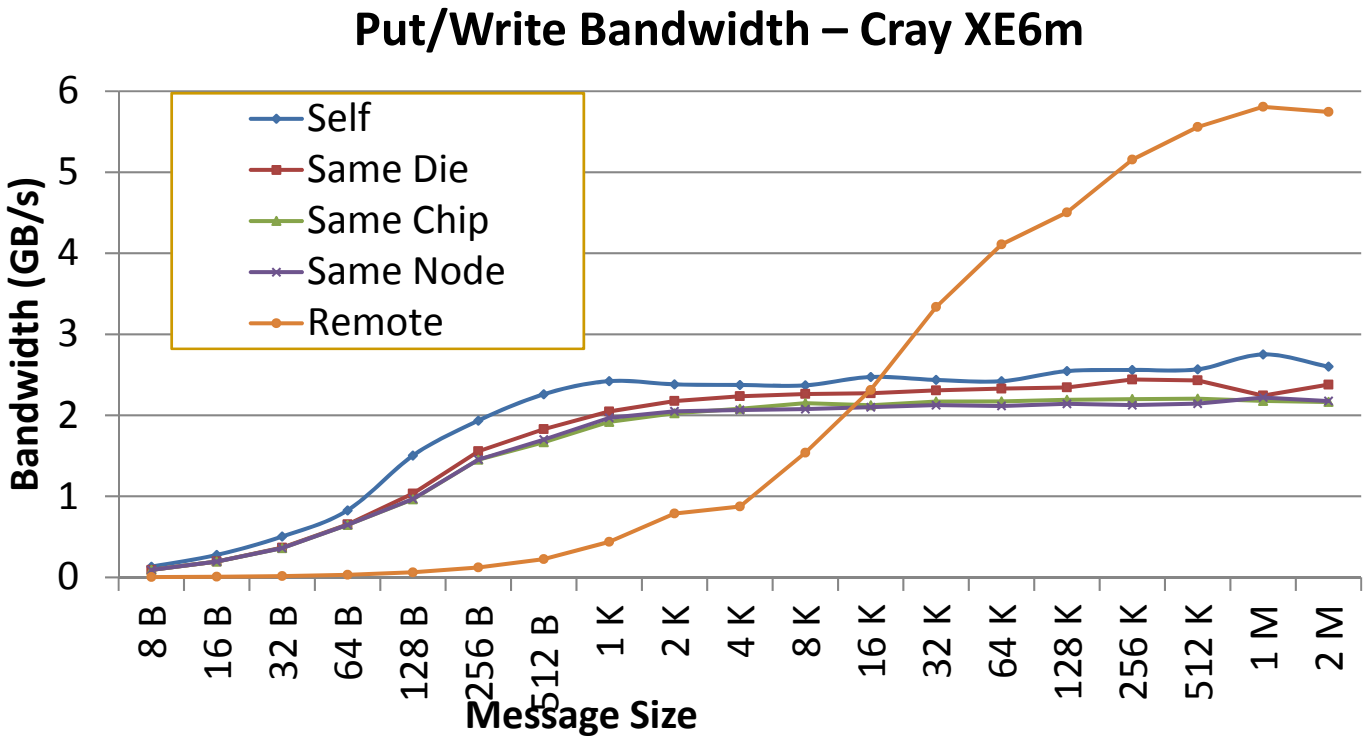
- Proper placement will
  - Avoid unnecessary data movement by exploiting locality
  - Utilize the shared memory and caches in the neighborhood
  - Utilize the best interconnect for the underlying communication
  - Yield a rising benefit as the size of your system increases! A must for exascale!!

Effect of Exploiting Hier Locality (Read Access)



Synthetic benchmark showing the gain of proper with varying number of threads and percentage of remote communication

Motivations and Early Investigations

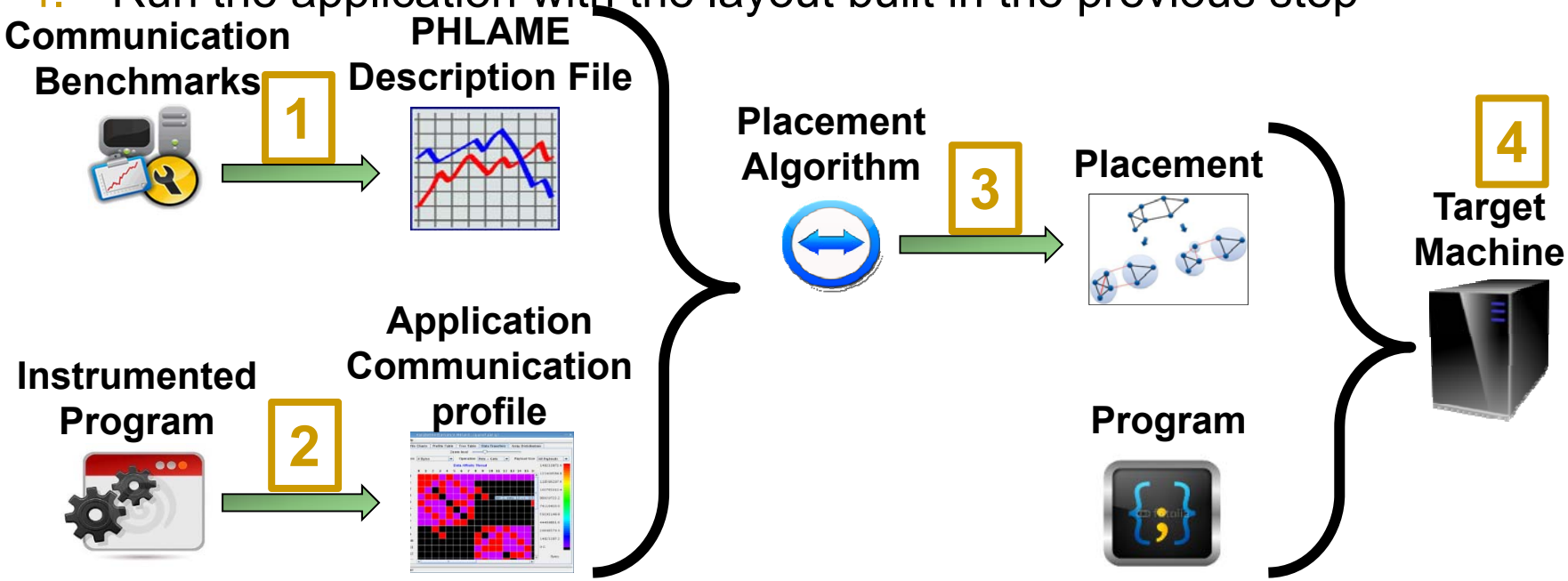


- The response of each level to communication varies according to message sizes
  - Closer is not always faster
    - Know and characterize your architecture!!

PHLAME Methodology

(Parallel Hierarchical Abstraction Model of Execution)

1. Characterize the machine message costs at each level to generate PHLAME description File (PDF)
2. Profile the application communication
3. Build a placement layout for the threads based on the above
4. Run the application with the layout built in the previous step



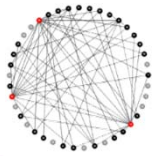
1 Characterizing the target machine

- ***Message cost:*** *total time for message to be delivered*

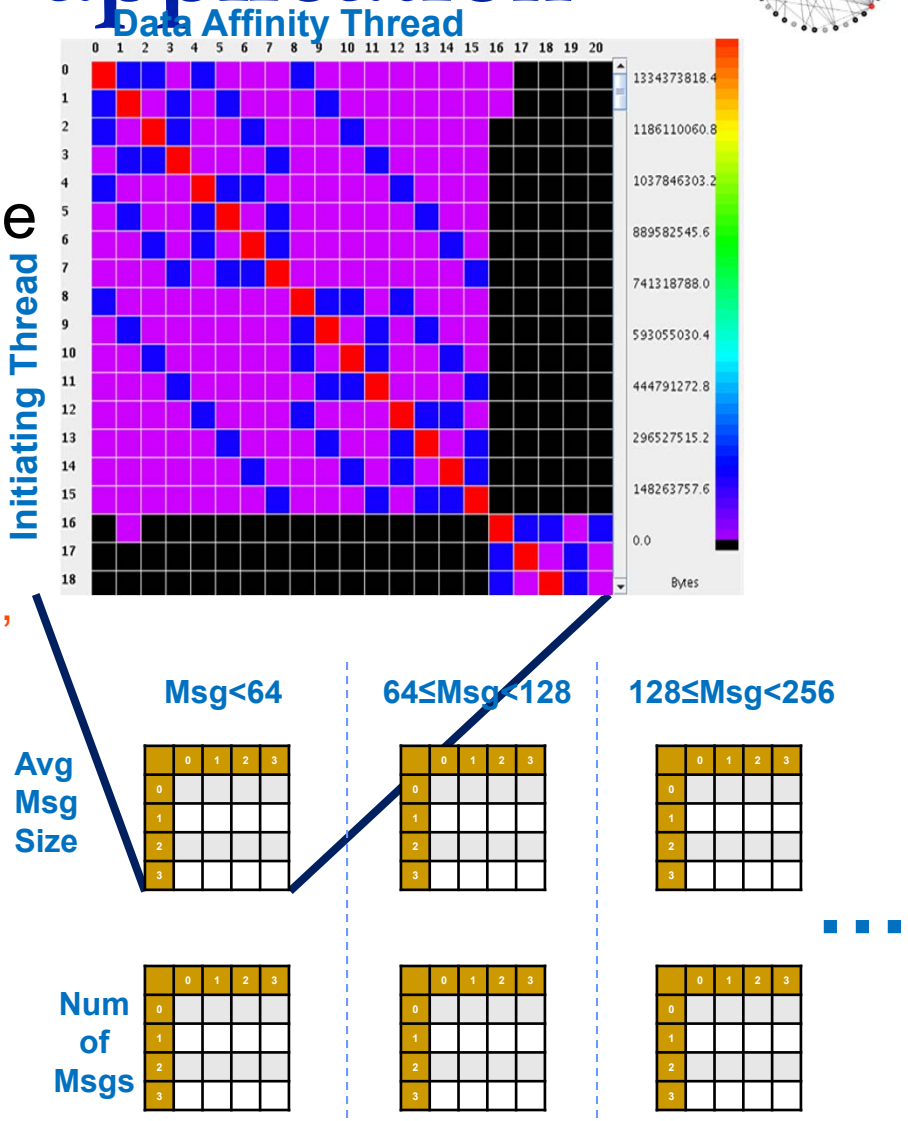
Level \ Msg(bytes)	1	8	16	32	64	128	…
1	0.516956	0.665469	1.209482	1.986097	3.606203	7.593014	
2	0.688468	1.038422	1.54703	2.772387	5.138746	10.86957	
3	0.687853	1.033378	1.543448	2.770083	5.128205	10.85776	
4	0.706414	1.05042	1.548707	2.77855	5.128205	11.02536	

Example: time per message (ns) machine communication characterization

2 Characterizing the application communication



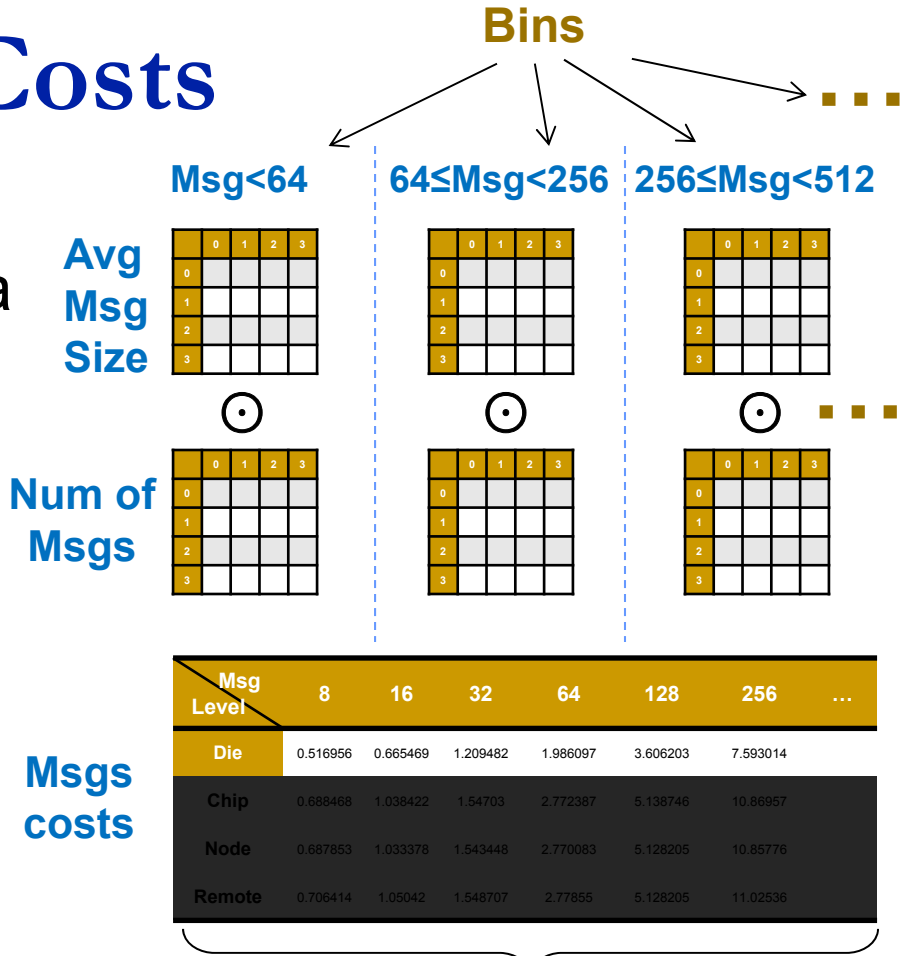
- Instrument the application code
  - generate the communication activity matrices
- The message sizes range is partitioned into bins
  - Each bin corresponds to a sub range, example: 1→64, 64→128, …
- There are two communication activity matrices for each bin
  - Average size
  - Number of messages



3 Calculating Level Costs

- Placement decisions require a measurement of how threads ***fit*** together

- Repeat for each level l :
  - For each pair of threads (i, j), where $i \neq j$, calculate the cost of their communication



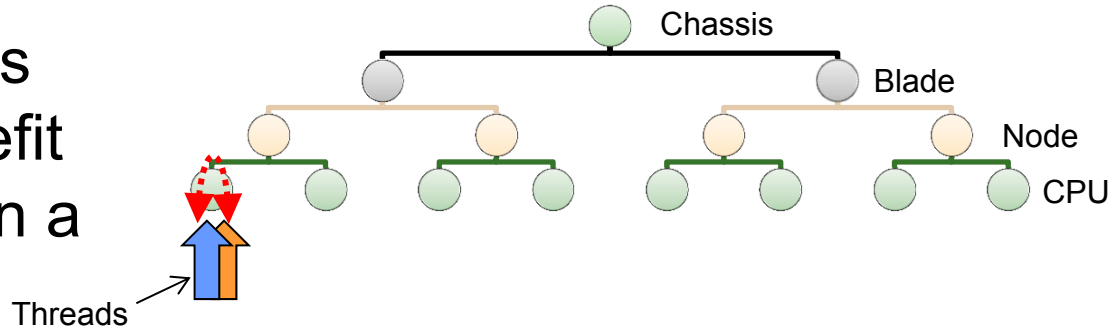
$$LevelCost_{lij} = \sum_{b=1}^{B} (AvgMsgSize_{ijb} \times NumMsgs_{ijb} \times LevelBinCost_{lb})$$ =

Level l costs

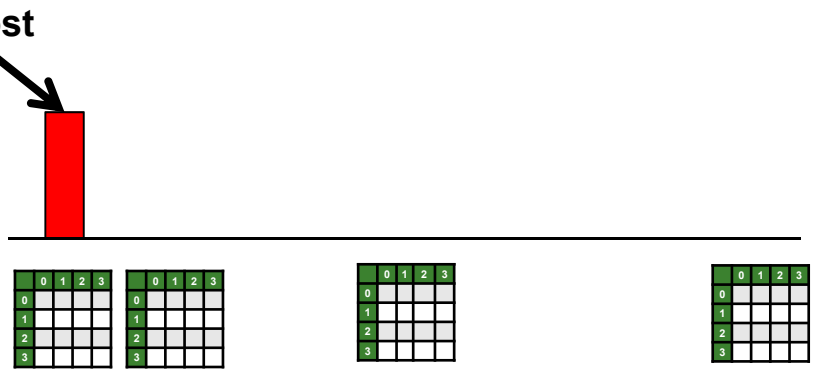
Where B is the number of bins

Hierarchical Thread Fitness Measure

- The fit measure shows how two threads benefit or lose if scheduled on a given level



- The fit measure is based on the difference of message costs at each level



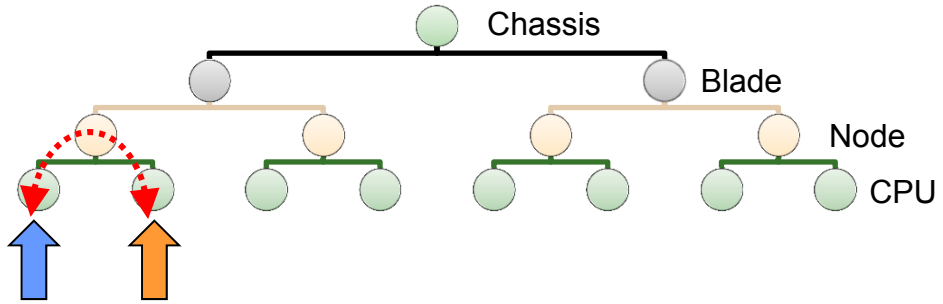
Gain of placement at current level, given current level is better

loss due to placement at current level, given current level is worse

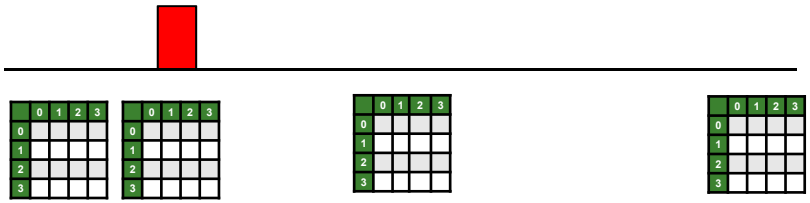
$$FIT\ Measure(i,j,L) = \sum (Worse - Level) - \sum (Level - Better)$$

Hierarchical Thread Fitness Measure

- The fit measure shows how two threads benefit or lose if scheduled on a given level



- The fit measure is based on the difference of message costs at each level



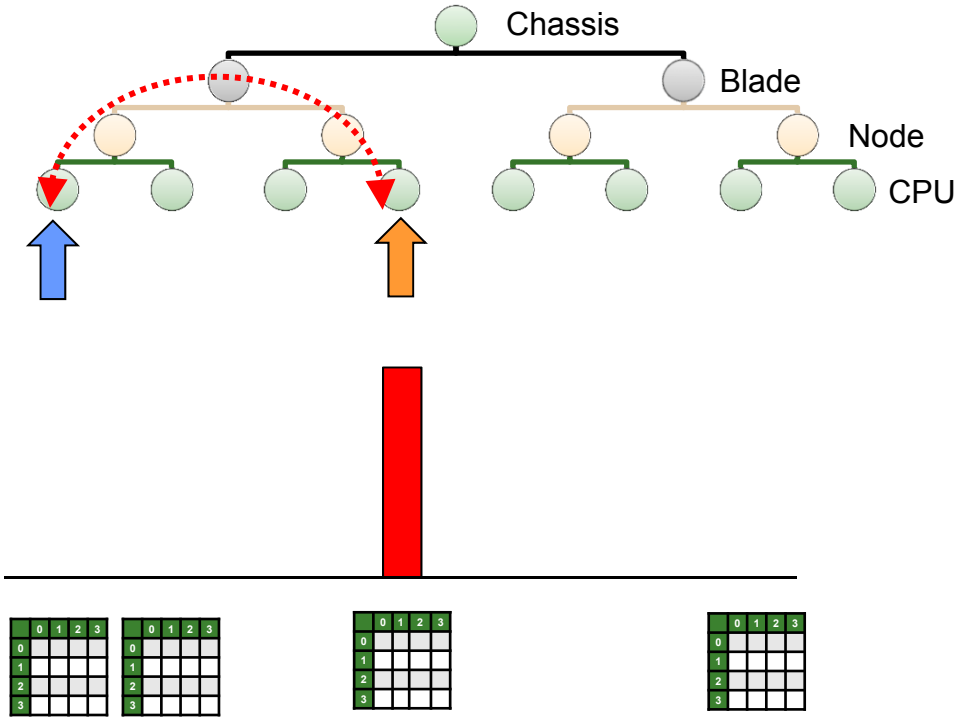
Gain of placement at current level, given current level is better

loss due to placement at current level, given current level is worse

$$FIT\ Measure(i,j,L) = \sum (Worse - Level) - \sum (Level - Better)$$

Hierarchical Thread Fitness Measure

- The fit measure shows how two threads benefit or lose if scheduled on a given level
- The fit measure is based on the difference of message costs at each level



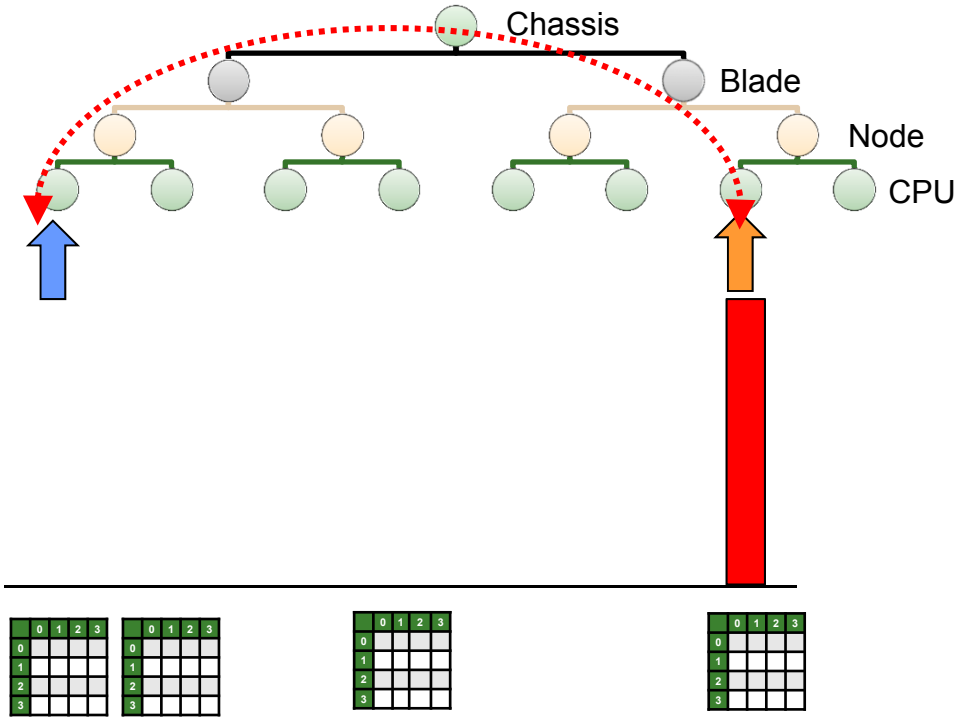
Gain of placement at current level, given current level is better

loss due to placement at current level, given current level is worse

$$FIT\ Measure(l,i,j) = \sum (Worse - Level) - \sum (Level - Better)$$

Hierarchical Thread Fitness Measure

- The fit measure shows how two threads benefit or lose if scheduled on a given level
- The fit measure is based on the difference of message costs at each level



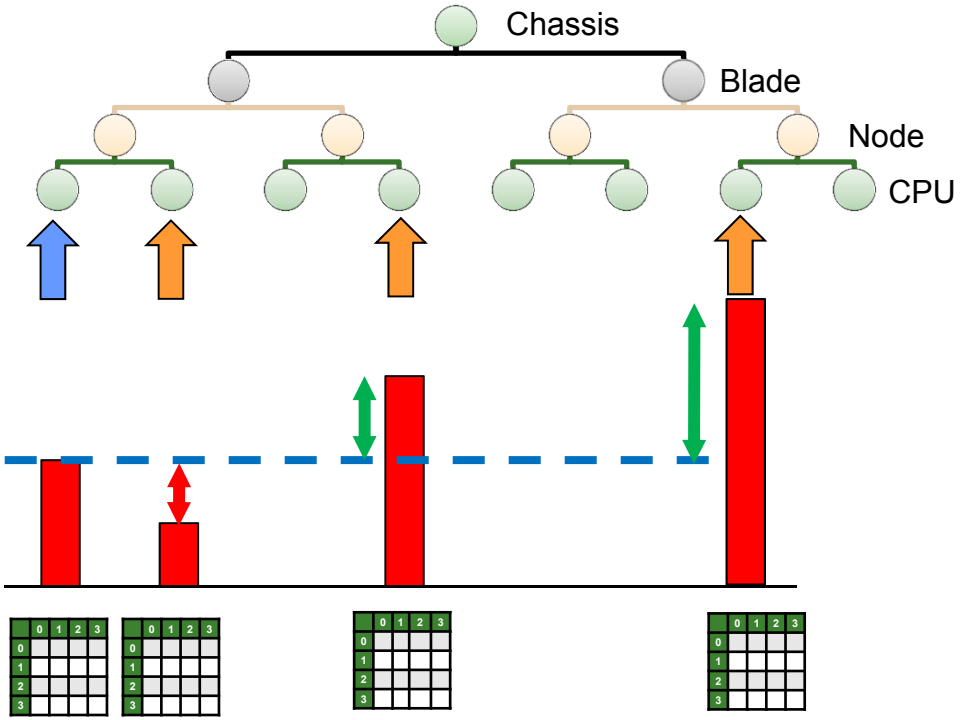
Gain of placement at current level, given current level is better

loss due to placement at current level, given current level is worse

$$FIT\ Measure(i,j,L) = \sum (Worse - Level) - \sum (Level - Better)$$

Hierarchical Thread Fitness Measure

- The fit measure shows how two threads benefit or lose if scheduled on a given level
- The fit measure is based on the difference of message costs at each level

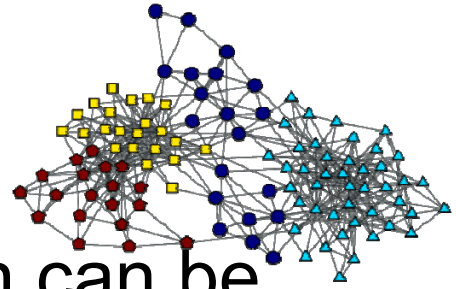


Gain of placement at current level, given current level is better

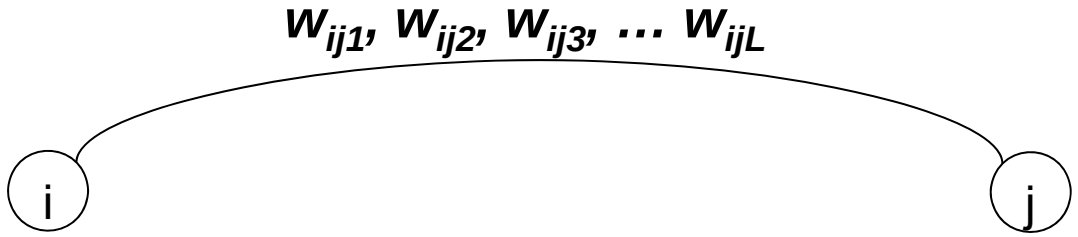
loss due to placement at current level, given current level is worse

$$FIT\ Measure(i,j,L) = \sum (Worse - Level) - \sum (Level - Better)$$

Mapping to Graph Theory

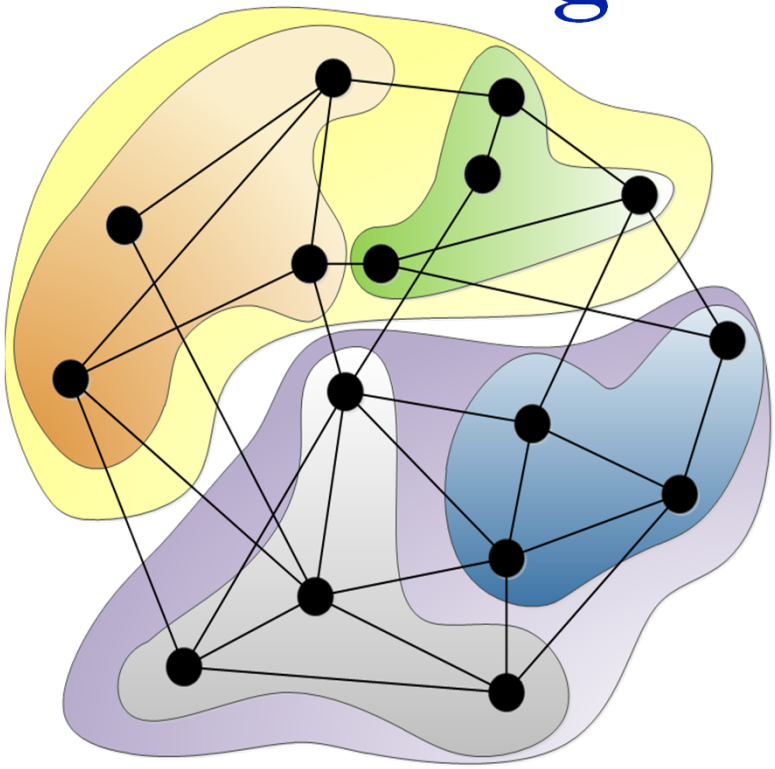


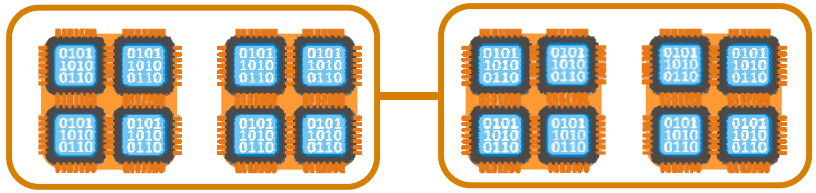
- The application communication pattern can be mapped into a graph
  - Vertices represent the threads
  - Edges represent interactions between threads
  - HTF at each level are edge weights
    - Multiple weights per edge



Hierarchical Graph Partitioning

- Algorithms can be
  - Bottom Up
    - Form partitions at lower levels first and recursively group them at higher levels
  - Top Down
    - Form partitions at upper levels first and recursively break them at lower levels





Abstract Machine:
$Level_1$: Width = 4 (# of locales)
MaxLocaleSize = 4 (# of cores in each locale)

Testbed

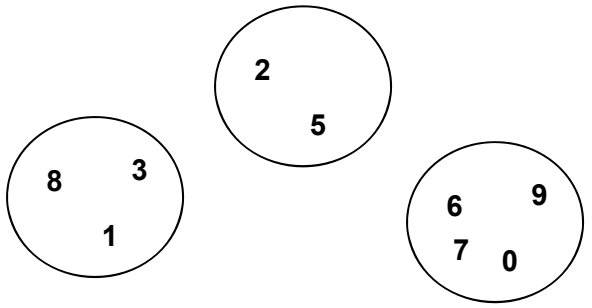
- Cray XE6m/XK7m
  - 24 cores per node
    - Two 12-core AMD Magny Cours
  - Gemini Interconnect
    - 2D Torus
- UPC NPB Benchmark from GWU
  - IS – Class C
  - FT – Class C
  - CG – Class C
  - MG – Class C
  - EP – Class C
- Heat Diffusion

Profiling the application communication – Implementation

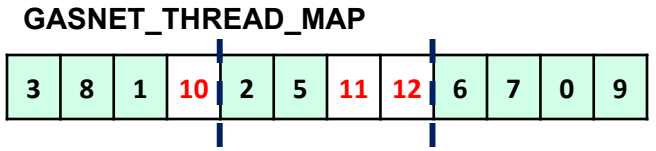
- TAU is selected to profile UPC and MPI programs
  - Generates activity matrix for each bin
- Bins are not supported in TAU profiles
- Modifications were made to TAU backend and frontends to support bins

Customizing GASNet

- The clustering algorithm usually assigns unequal number of threads to different nodes



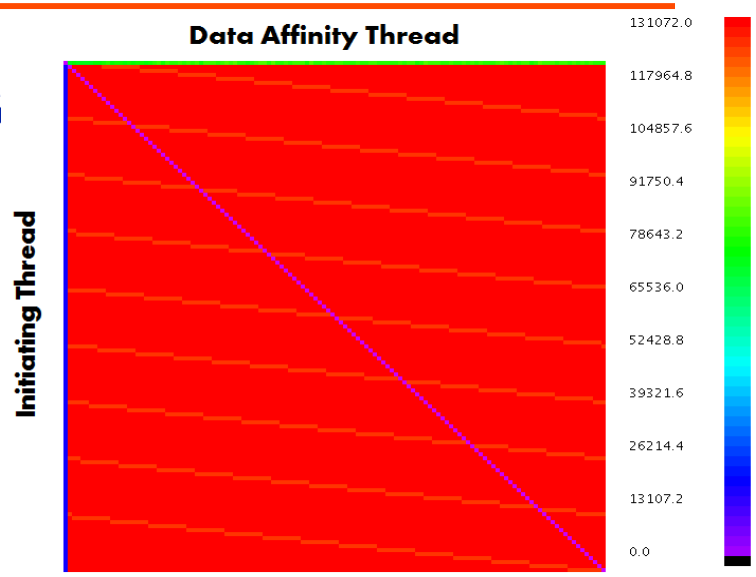
- The Cray Application Level Placement Scheduler (ALPS) does not support this feature

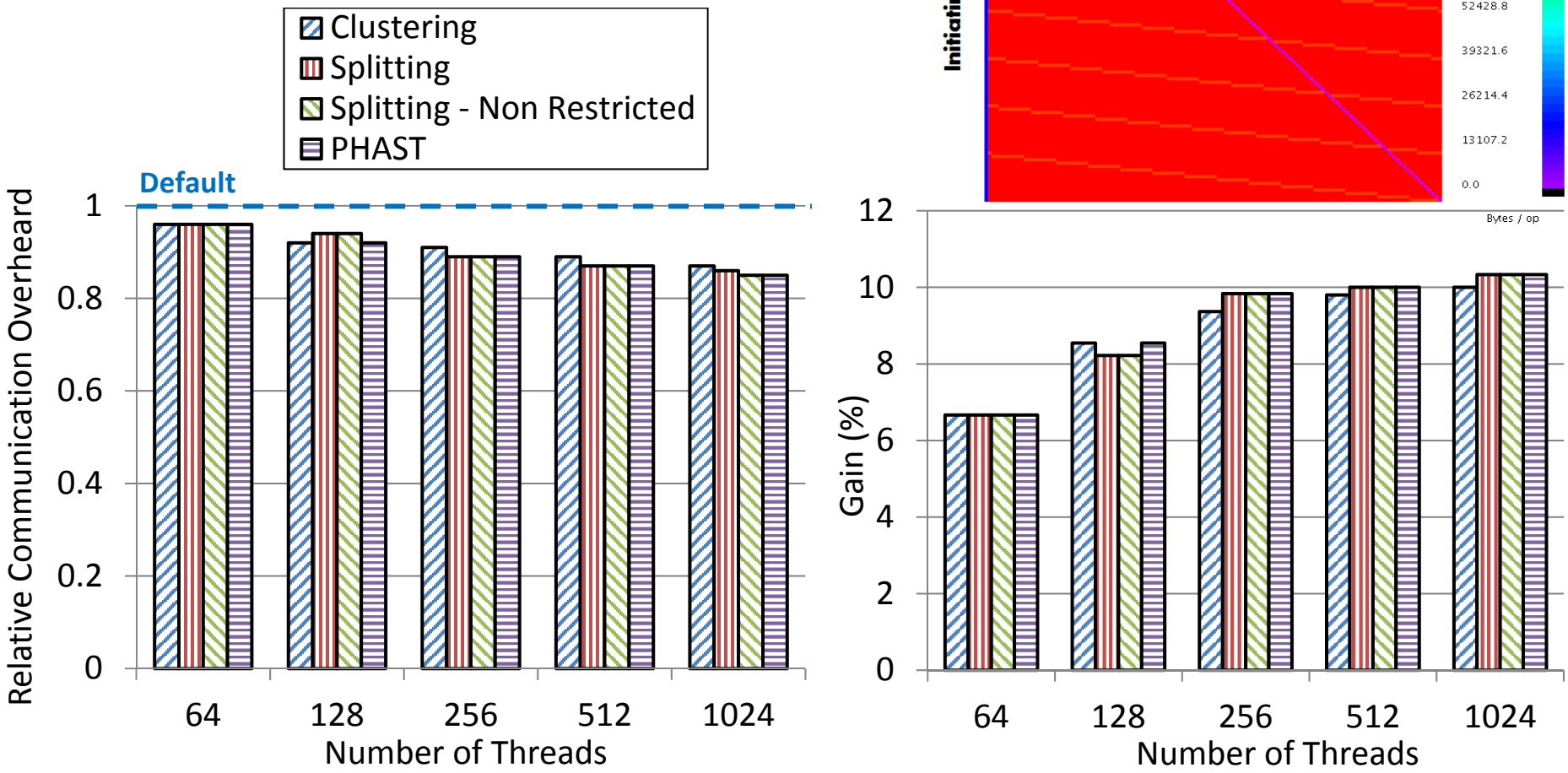


- A modified GASNet Geminie Conduit was used to trick the system to achieve the non-uniform thread count per node
  - Dummy processes are launched
  - Environment variables control how the runtime pick the correct number of processes on each node

Experimental Results

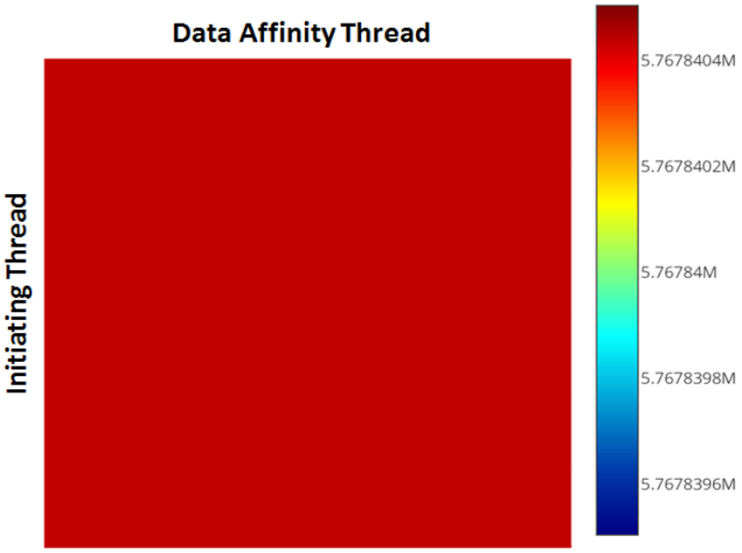
- FT – all-to-all communication

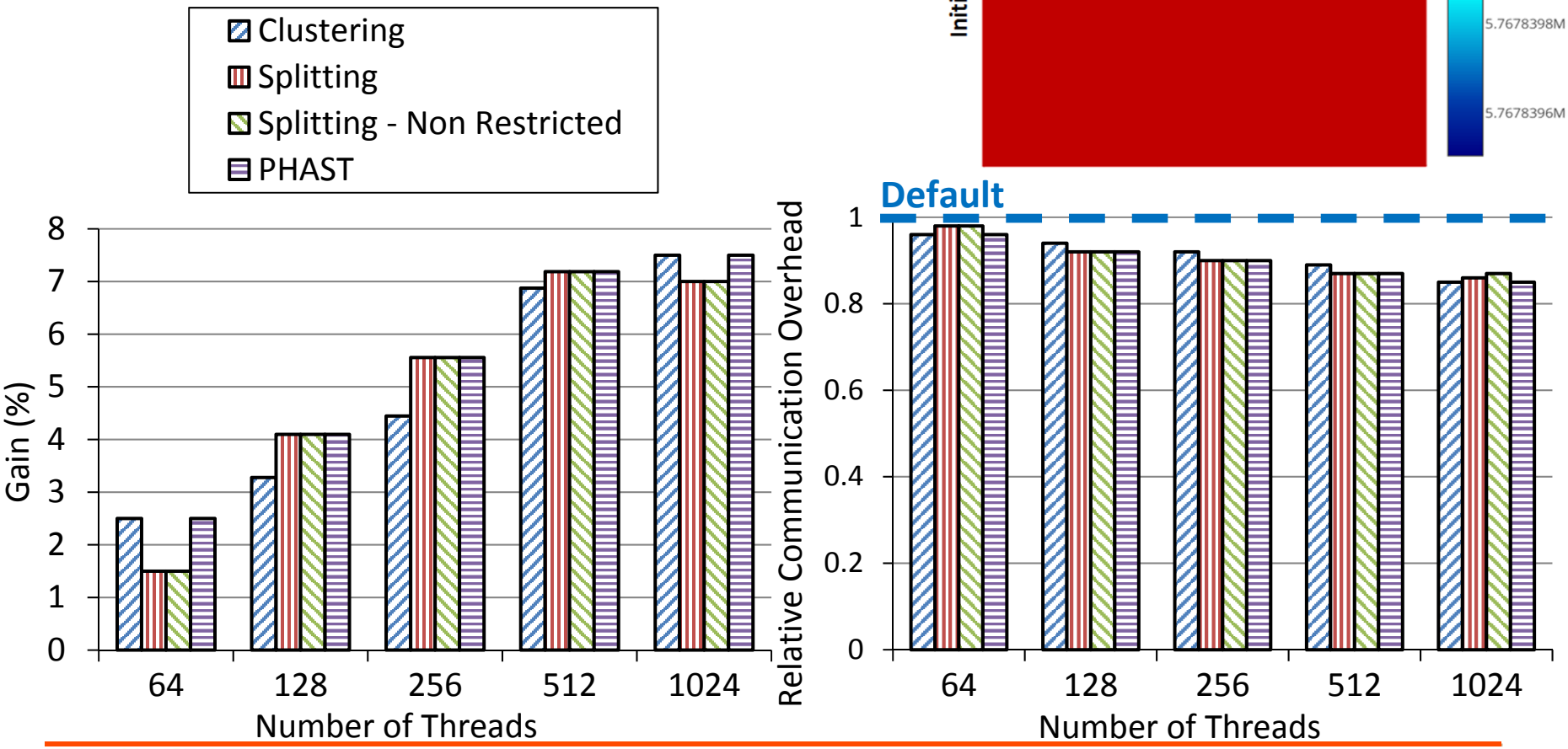




Experimental Results MPI

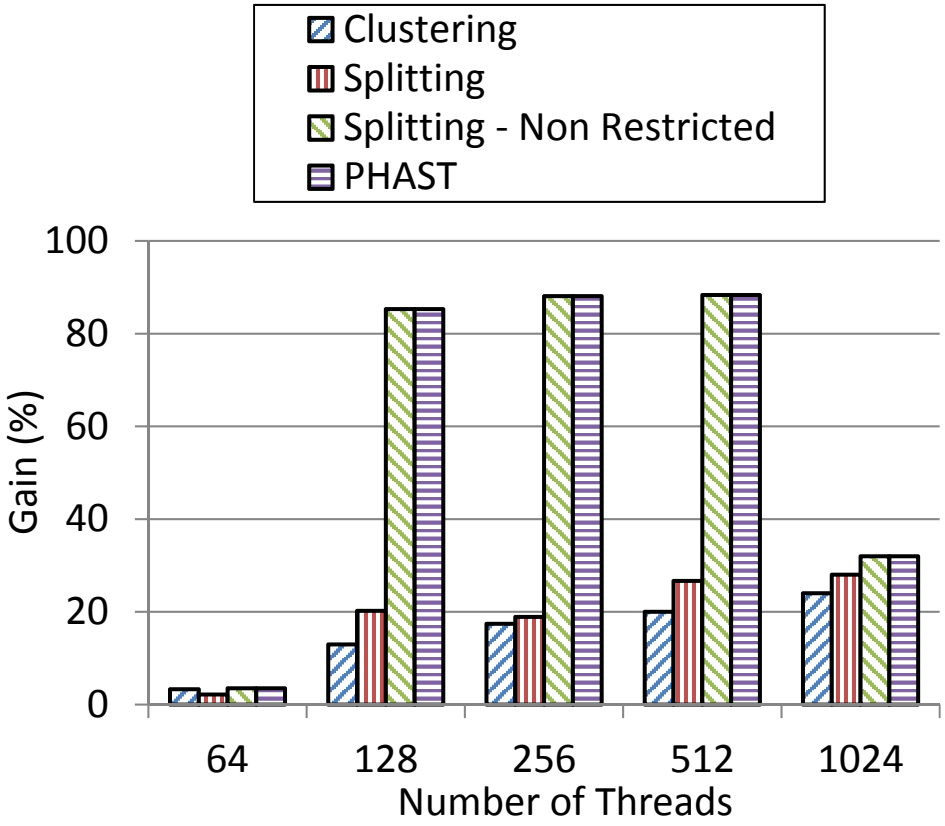
- FT – all-to-all communication

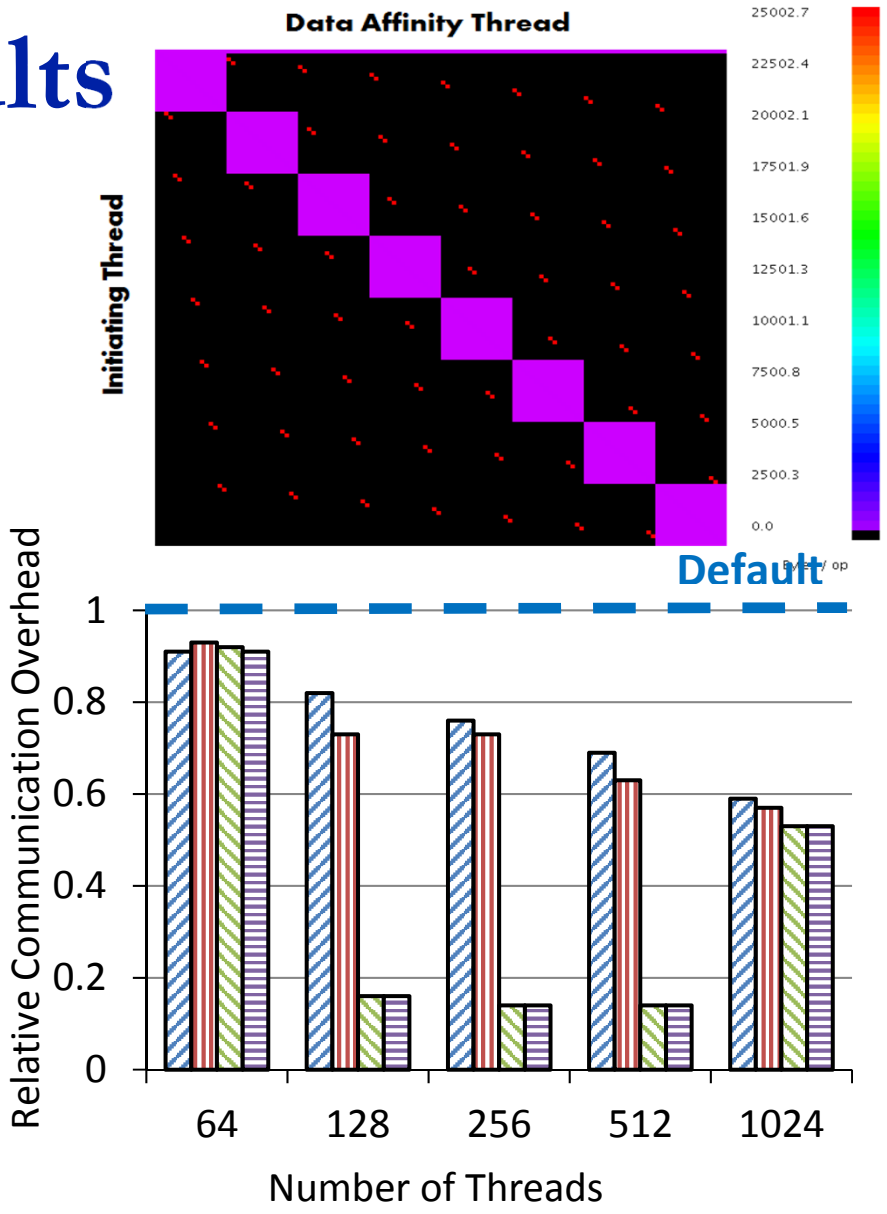




Experimental Results UPC

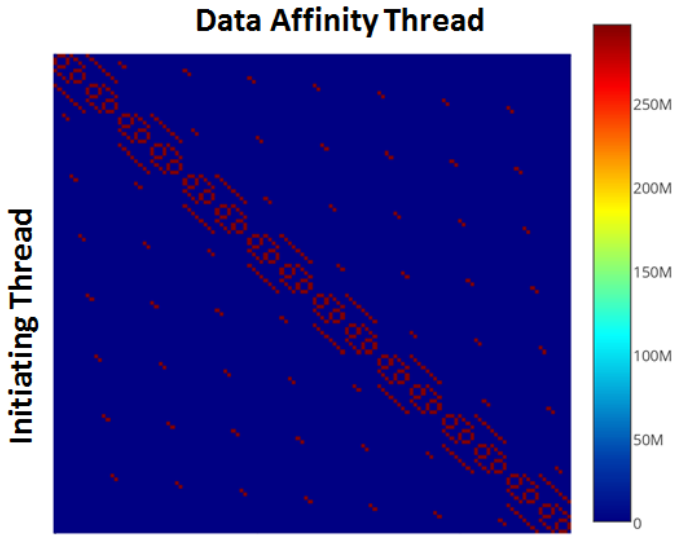
- CG – Irregular memory access and communication

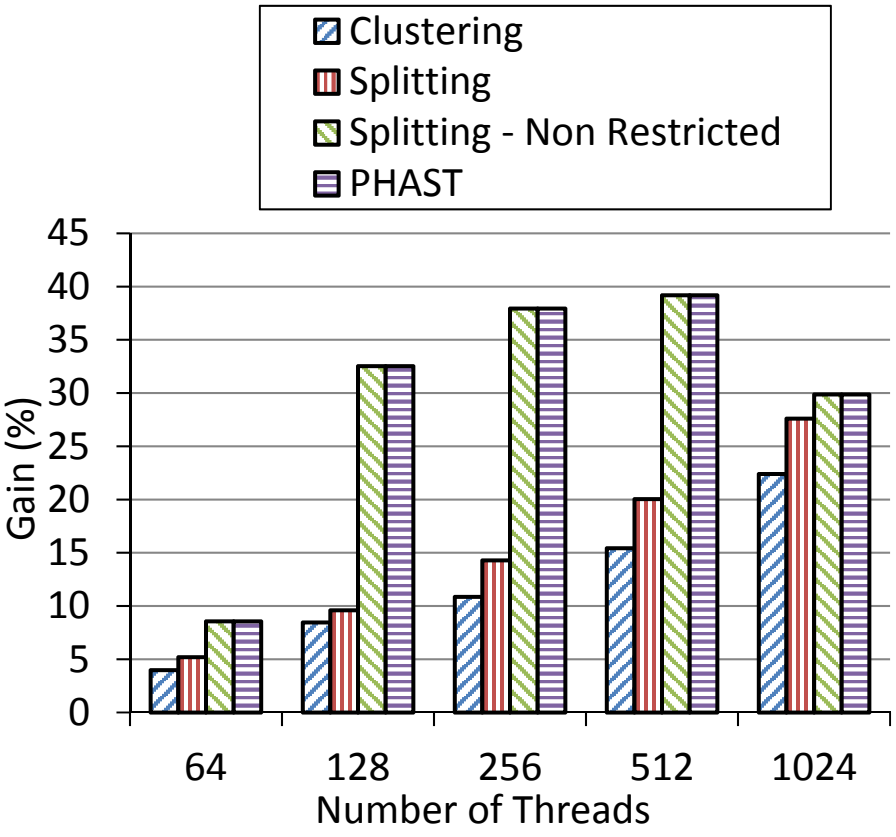


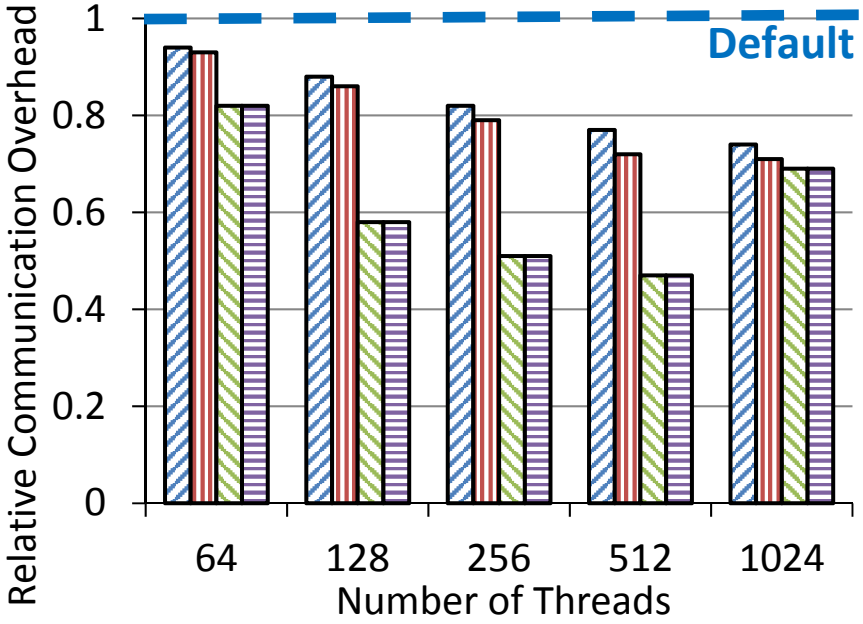


Experimental Results MPI

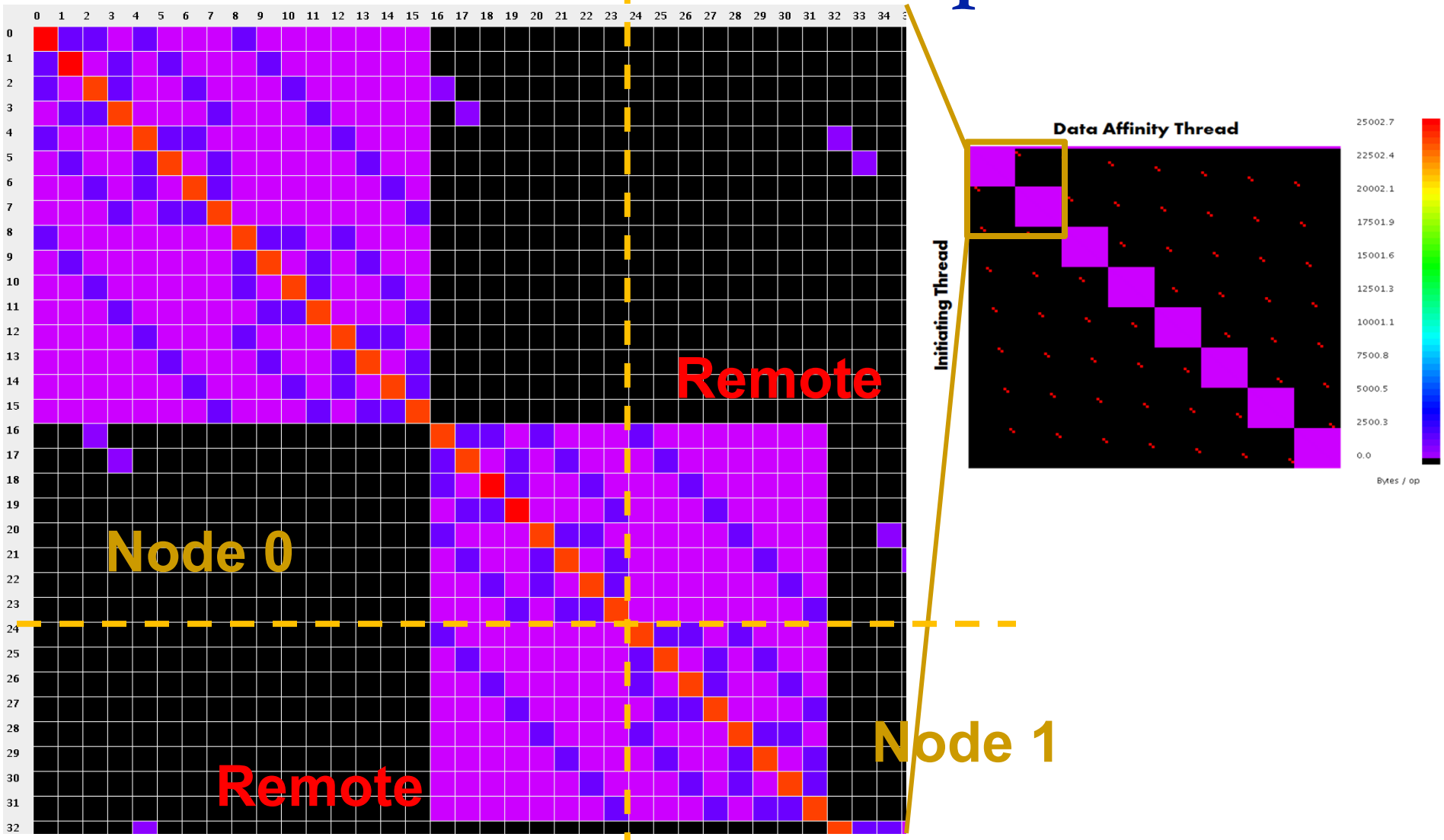
- CG – Irregular memory access and communication







CG – Non Restricted Explanation



Overview

- Fundamental Challenges for Extreme Computing
- Locality and Hierarchical Locality
- Programming Models
- Hardware Support for Productive Locality Exploitation- Address Remapping
- Hierarchical Locality Exploitation
- **Concluding Remarks**

Concluding Remarks

- Due to energy and bandwidth constrains data movements are becoming too expensive
- Locality exploitation is an obvious target
- Extreme scale architectures are becoming deeply hierarchical giving rise to hierarchical locality
- Hierarchical locality exploitation must be done productively, leaving programmers with the necessary min work to do
- We can expect some programming paradigms to provide explicit solutions
- Locality-aware programming, hardware support and run-time systems can play a bigger role while keeping programmers productivity

Publications

- Ahmad Anbar, Olivier Serres, Engin Kayraklioglu, Abdel Hamid Badawy, and Tarek El-Ghazawi, "Exploiting Hierarchical Locality in Deep Parallel Architectures". ACM Transactions on Architecture and Code Optimizations. Volume 13 Issue 2, June 2016 .
- Olivier Serres, Abdullah Kayi, Ahmed Anbar and Tarek El-Ghazawi, "Enabling PGAS Productivity with Hardware Support for Shared Address Mapping: A UPC Case Study". ACM Transactions on Architecture and Code Optimizations. Volume 12 Issue 4, January 2016.
- Ahmad Anbar, Abdel-Hameed Badawy, Olivier Serres and Tarek El-Ghazawi, "Where Should The Threads Go? Leveraging Hierarchical Data Locality to Solve the Thread Affinity Dilemma," in Proc. 20th International Conference on Parallel and Distributed Systems (ICPADS 2014). IEEE, Hsinchu, Taiwan, Dec 16-19, 2014.
- Ahmad Anbar, Olivier Serres, Engin Kayraklioglu, Abdel-Hameed Badawy , Tarek El-Ghazawi PHLAME: Hierarchical Locality Exploitation Using the PGAS Model. IEEE International Conference on Partitioned Global Address Space Programming Models (PGAS 2015), Washington DC, September 18-20, 2015.

3. Olivier Serres, Abdullah Kayi, Ahmad Anbar, and Tarek El-Ghazawi, "Enabling PGAS productivity with hardware support for shared address mapping; a UPC case study," in Proc. 16th IEEE International Conference on High Performance Computing and Communications, August 20-22, 2014.

Follow up work in Hierarchical Locality Exploitation

- Use thread data-affinity from locality-aware program as a starting point into a hierarchical locality exploitation system (PHLAME or FLAME: Parallel Hierarchical Abstraction Model of Execution)
- Examine best graph partitioning methods
- Decentralize algorithms, and build in fast predictions to handle the Exascale
- Consider dynamic solutions
- Consider unprofiled cases and collecting intelligence on runs for later use and optimizations
- Consider data dependent cases
- Consider dynamic parallelism cases
- Investigate hardware support