The HPC Challenge (HPCC) Benchmark Suite

Piotr Luszczek, David Bailey,
Jack Dongarra, Jeremy Kepner,
Robert Lucas, Rolf Rabenseifner,
Daisuke Takahashi

SC06, Tampa, Florida
Sunday, November 12, 2006
Session S12
1:30-5:00

Acknowledgements

- This work was supported in part by the Defense Advanced Research Projects Agency (DARPA), the Department of Defense, the National Science Foundation (NSF), and the Department of Energy (DOE) through the DARPA High Productivity Computing Systems (HPCS) program under grant FA8750-04-1-0219 and under Army Contract W15P7T-05-C-D001
- Opinions, interpretations, conclusions, and recommendations are those of the authors and are not necessarily endorsed by the United States Government

Introduction

- **HPC Challenge Benchmark Suite**
  - **To examine the performance of HPC architectures using kernels with more *challenging* memory access patterns than HPL**
  - **To *augment* the TOP500 list**
  - **To provide benchmarks that *bound* the performance of many real applications as a function of memory access characteristics — e.g., spatial and temporal locality**
  - **To outlive HPCS**
- **HPC Challenge pushes spatial and temporal boundaries and defines performance bounds**

TOP500 and HPCC

- TOP500
  - Performance is represented by only a single metric
  - Data is available for an extended time period (1993-2005)
- Problem:
  There can only be one "*winner*"
- Additional metrics and statistics
  - Count (single) vendor systems on each list
  - Count total flops on each list per vendor
  - Use external metrics: price, ownership cost, power, …
  - Focus on growth trends over time

- HPCC
  - Performance is represented by multiple single metrics
  - Benchmark is new — so data is available for a limited time period (2003-2005)
- Problem:
  There cannot be one "*winner*"
- We avoid "*composite*" benchmarks
  - Perform trend analysis
    - HPCC can be used to show complicated kernel/ architecture performance characterizations
  - Select some numbers for comparison
  - Use of kiviat charts
    - Best when showing the differences due to a single independent "variable"
- Over time — also focus on growth trends

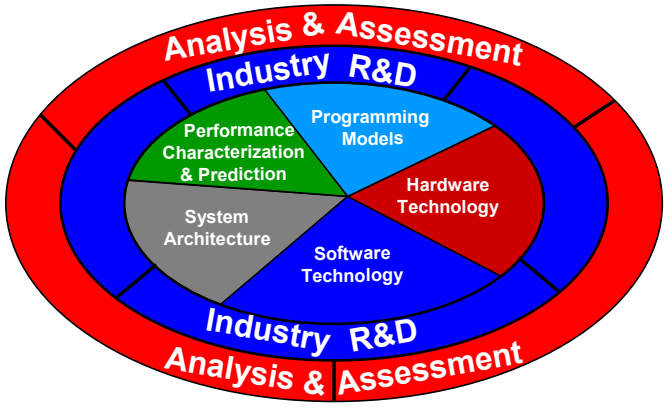
High Productivity Computing Systems (HPCS)

Goal:

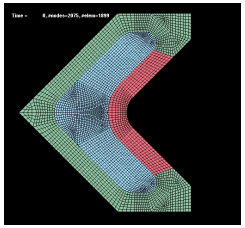
- **Provide a new generation of economically viable high productivity computing systems for the national security and industrial user community (2010)**

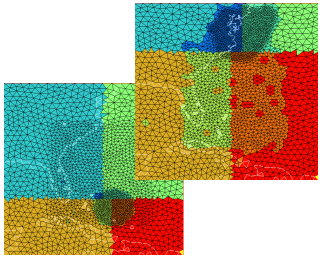
Impact:

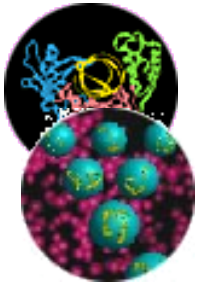
- **Performance** (time-to-solution): speedup critical national security applications by a factor of 10X to 40X
- **Programmability** (idea-to-first-solution): reduce cost and time of developing application solutions
- **Portability** (transparency): insulate research and operational application software from system
- **Robustness** (reliability): apply all known techniques to **protect against outside attacks**, hardware faults, & programming errors



HPCS Program Focus Areas





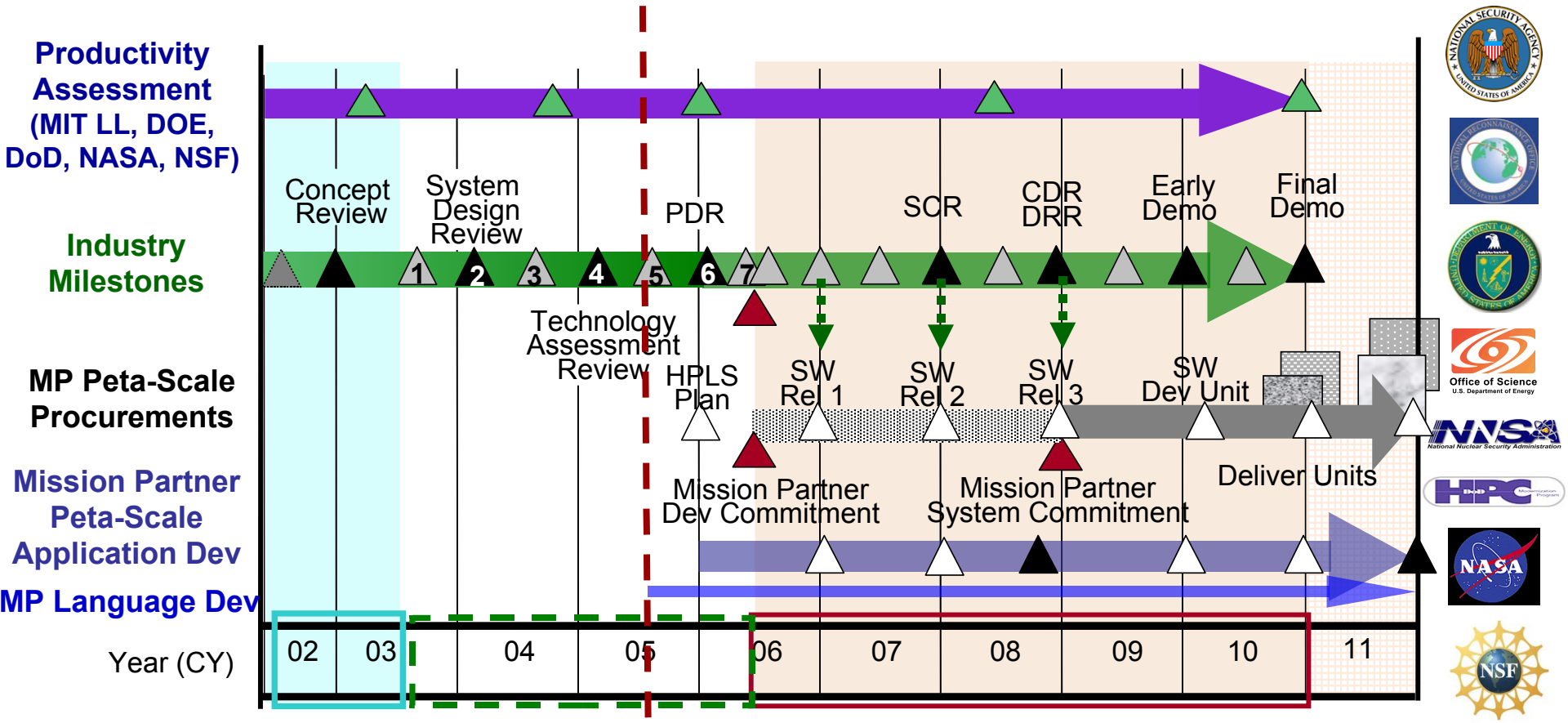


Applications:

- Intelligence/surveillance, reconnaissance, cryptanalysis, weapons analysis, airborne contaminant modeling and biotechnology

Fill the Critical Technology and Capability Gap
Today (late 80's HPC technology).....*to*.....Future (Quantum/Bio Computing)

HPCS Program Phases I-III



Program Reviews

Critical Milestones

Program Procurements

(Funded Five) Phase I Industry Concept Study

(Funded Three) Phase II R&D

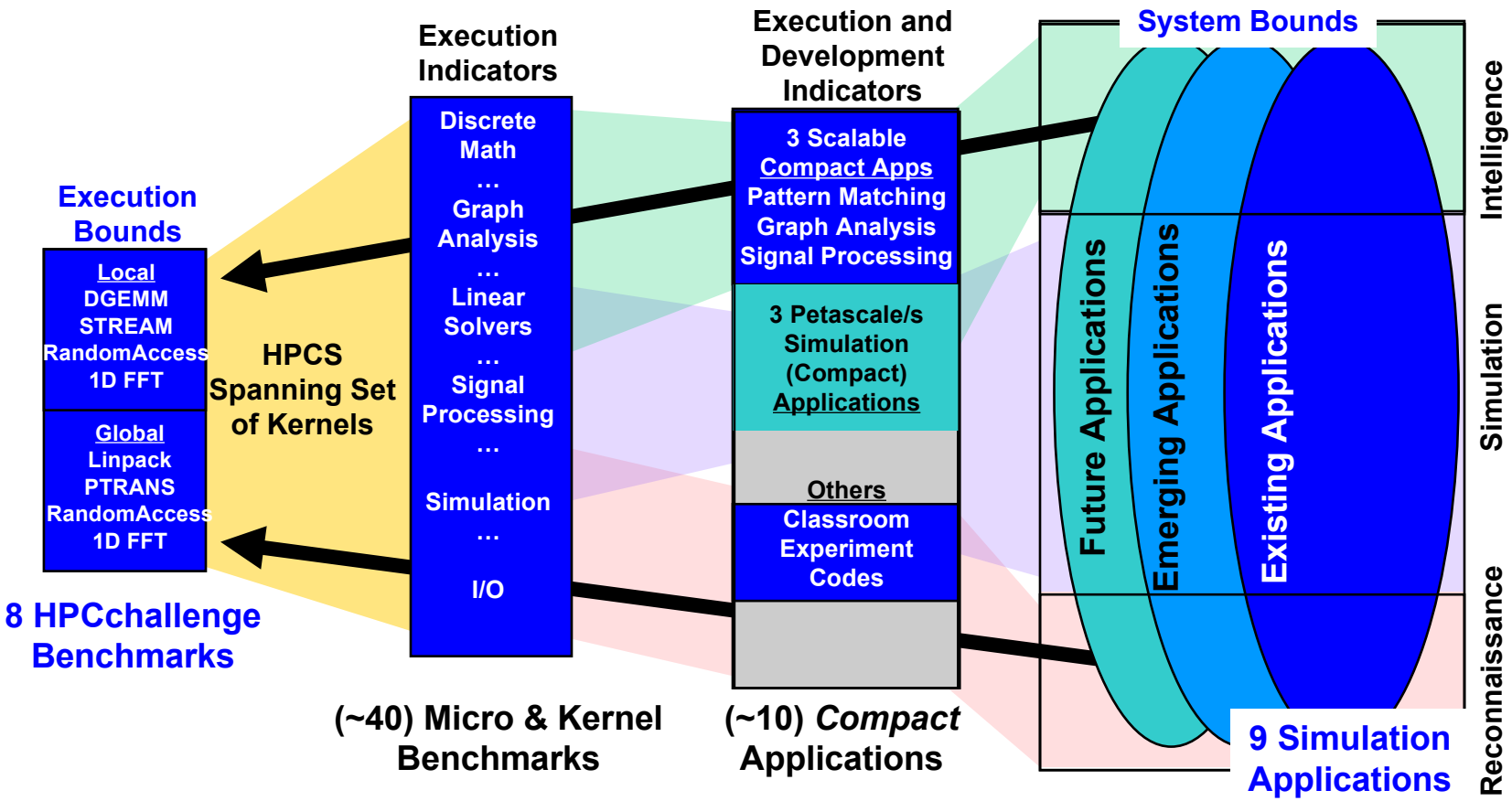
Phase III Prototype Development

Mission Partners

CRAY IBM Sun microsystems

Council on Competitiveness

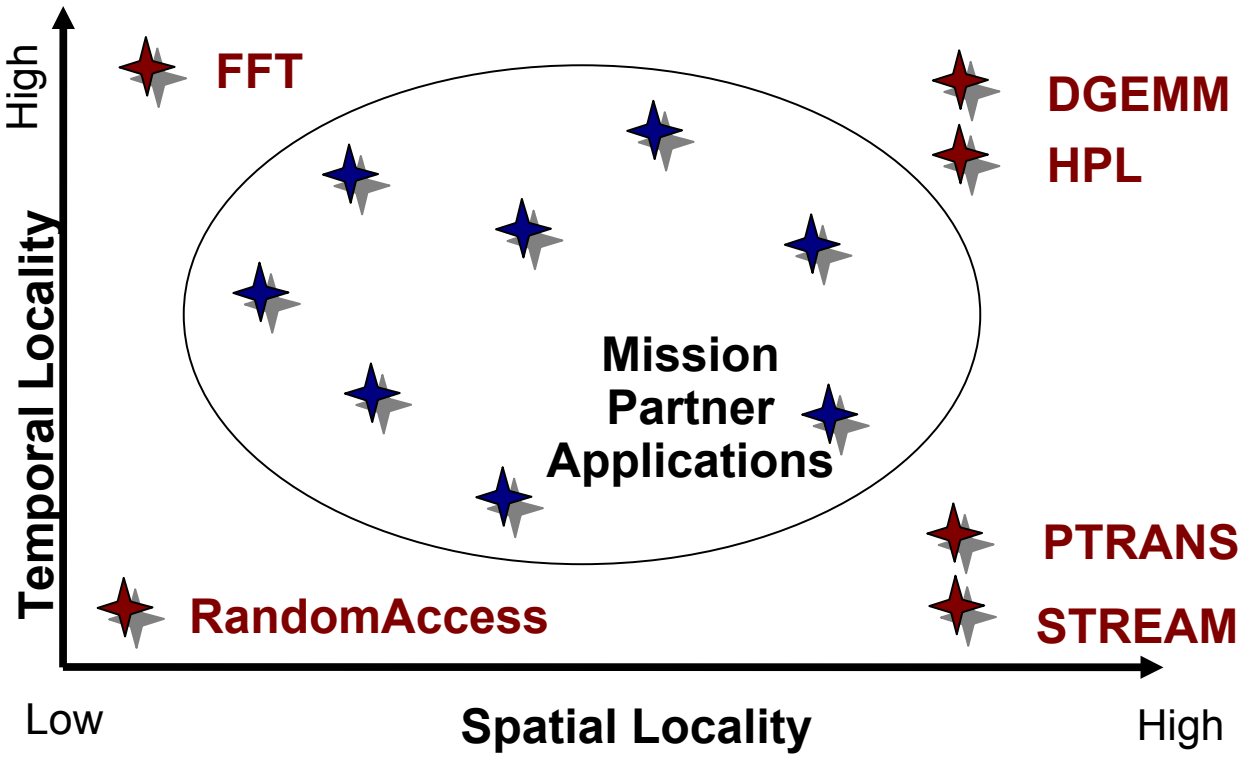
HPCS Benchmark and Application Spectrum



Spectrum of benchmarks provide different views of system

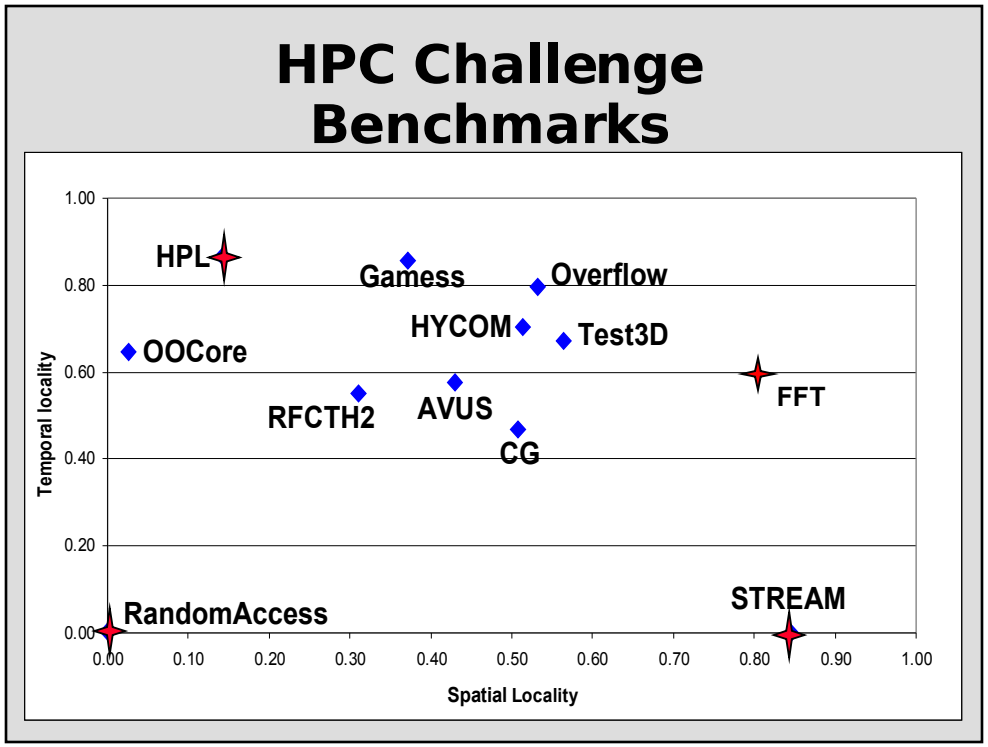
- HPCchallenge pushes spatial and temporal boundaries; sets performance bounds
- Applications drive system issues; set legacy code performance bounds
- Kernels and Compact Apps for deeper analysis of execution and development time

Motivation of the HPCC Design



Measuring Spatial and Temporal Locality (1/2)

Generated by PMaC @ SDSC



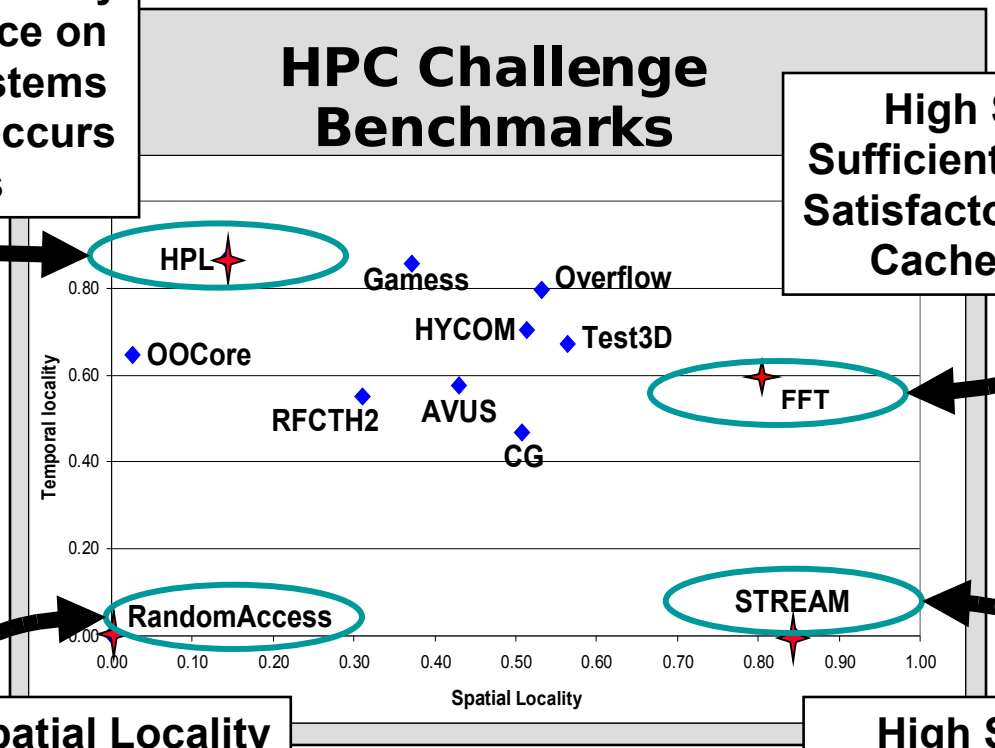
- Spatial and temporal data locality here is for one node/processor — i.e., locally or “in the small”

Measuring Spatial and Temporal Locality (2/2)

Generated by PMaC @ SDSC

High Temporal Locality
Good Performance on
Cache-based systems
Spatial Locality occurs
in registers

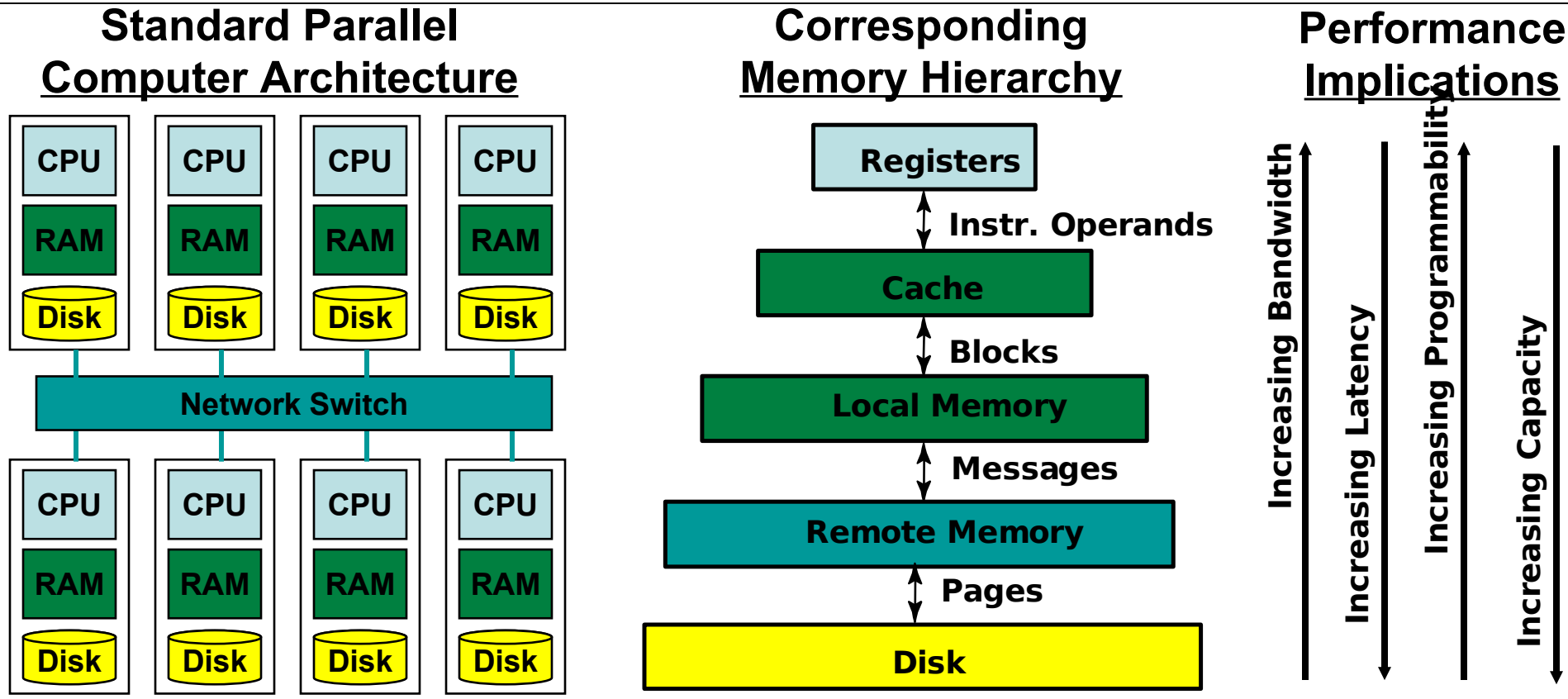
High Spatial Locality
Sufficient Temporal Locality
Satisfactory Performance on
Cache-based systems



No Temporal or Spatial Locality
Poor Performance on
Cache-based systems

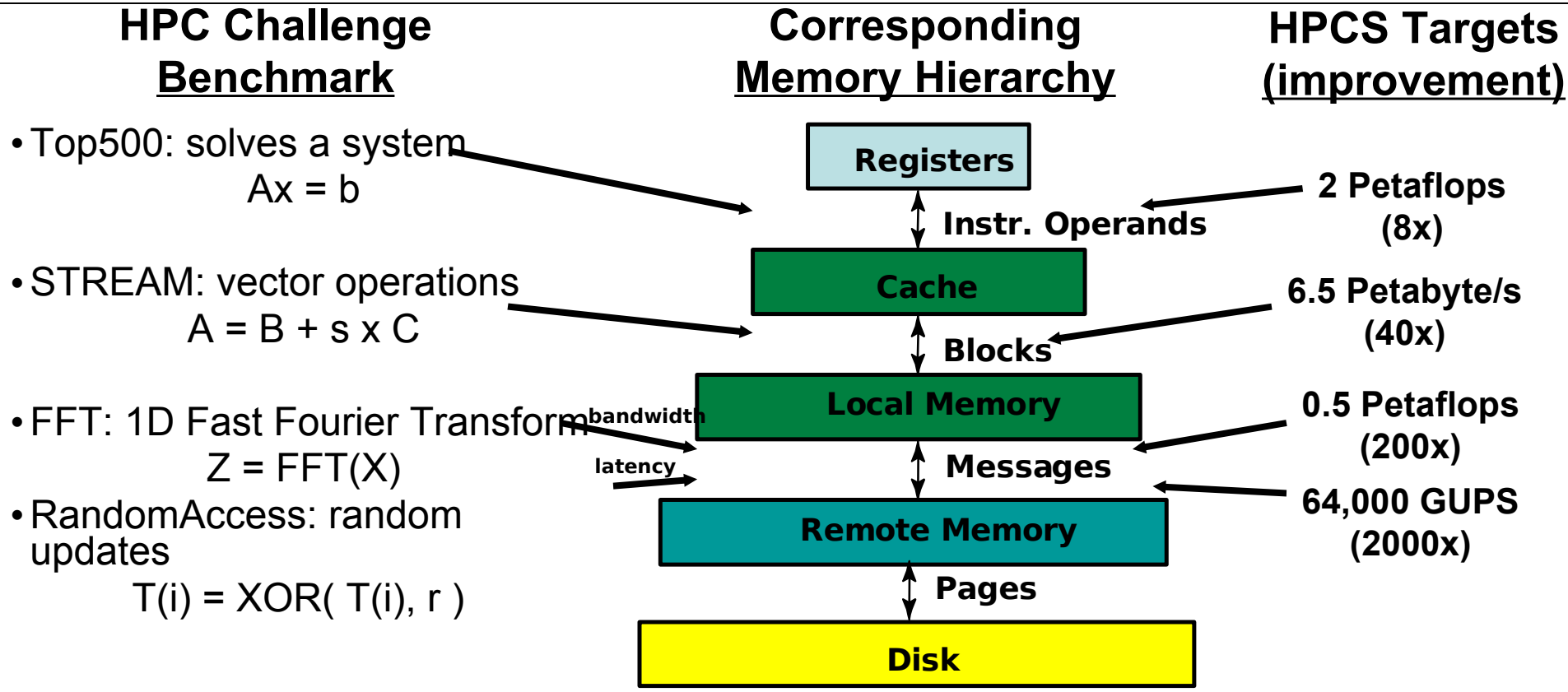
High Spatial Locality
Moderate Performance on
Cache-based systems

Supercomputing Architecture Issues



- Standard architecture produces a “steep” multi-layered memory hierarchy
  - Programmer must manage this hierarchy to get good performance
- HPCS technical goal
  - Produce a system with a “flatter” memory hierarchy that is easier to program

HPCS Performance Targets



- HPCS program has developed a new suite of benchmarks (HPC Challenge)
- Each benchmark focuses on a different part of the memory hierarchy
- HPCS program performance targets will flatten the memory hierarchy, improve real application performance, and make programming easier

Official HPCC Submission Process

1. Download
2. Install
3. Run
4. Upload results
5. Confirm via @email@
6. *Tune*
7. *Run*
8. *Upload results*
9. *Confirm via @email@*

Prequesites:
- C compiler
- BLAS
- MPI

Provide detailed installation and execution environment

- Only some routines can be replaced
- Data layout needs to be preserved
- Multiple languages can be used

Optional

Results are immediately available on the web site:
- Interactive HTML
- XML
- MS Excel
- Kiviat charts (radar plots)

HPCC as a Framework (1/2)

- Many of the component benchmarks were widely used before the HPC Challenge suite of Benchmarks was assembled
  - HPC Challenge has been more than a packaging effort
  - Almost all component benchmarks were augmented from their original form to provide consistent verification and reporting
- We stress the importance of running these benchmarks on a single machine — with a single configuration and options
  - The benchmarks were useful separately for the HPC community, meanwhile
  - The unified HPC Challenge framework creates an unprecedented view of performance characterization of a system
    - A comprehensive view with data captured the under the same conditions allows for a variety of analyses depending on end user needs

HPCC as a Framework (2/2)

- HPCC unifies a number of existing (and well known) codes in one consistent framework
- A single executable is built to run all of the components
  - Easy interaction with batch queues
  - All codes are run under the same OS conditions – just as an application would
    - No special mode (page size, etc.) for just one test (say Linpack benchmark)
    - Each test may still have its own set of compiler flags
      - Changing compiler flags in the same executable may inhibit inter-procedural optimization
- Why not use a script and a separate executable for each test?
  - Lack of enforced integration between components
    - Ensure reasonable data sizes
    - Either all tests pass and produce meaningful results or failure is reported
  - Running a single component of HPCC for testing is easy enough

Base vs. Optimized Run

- HPC Challenge encourages users to develop optimized benchmark codes that use architecture specific optimizations to demonstrate the best system performance
- Meanwhile, we are interested in both
  - The base run with the provided reference implementation
  - An optimized run
- The base run represents behavior of legacy code because
  - It is conservatively written using only widely available programming languages and libraries
  - It reflects a commonly used approach to parallel processing sometimes referred to as hierarchical parallelism that combines
    - Message Passing Interface (MPI)
    - OpenMP Threading
  - We recognize the limitations of the base run and hence we encourage optimized runs
- Optimizations may include alternative implementations in different programming languages using parallel environments available specifically on the tested system
- We require that the information about the changes made to the original code be submitted together with the benchmark results
  - We understand that full disclosure of optimization techniques may sometimes be impossible
  - We request at a minimum some guidance for the users that would like to use similar optimizations in their applications

HPCC Base Submission

- **Publicly available code is required for base submission**
  1. **Requires C compiler, MPI 1.1, and BLAS**
  2. **Source code cannot be changed for submission run**
  3. **Linked libraries have to be publicly available**
  4. **The code contains optimizations for contemporary hardware systems**
  5. **Algorithmic variants provided for performance portability**
- **This to mimic legacy applications' performance**
  1. **Reasonable software dependences**
  2. **Code cannot be changed due to complexity and maintenance cost**
  3. **Relies on publicly available software**
  4. **Some optimization has been done on various platforms**
  5. **Conditional compilation and runtime algorithm selection for performance tuning**

Baseline code has over 10k SLOC — there must more productive way of coding

HPCC Optimized Submission

- Timed portions of the code may be replaced with optimized code
- Verification code still has to pass
  - Must use the same data layout or pay the cost of redistribution
  - Must use sufficient precision to pass residual checks
- Allows to use new parallel programming technologies
  - New paradigms, e.g. one-sided communication of MPI-2:
    MPI_Win_create(…);
    MPI_Get(…);
    MPI_Put(…);
    MPI_Win_fence(…);
  - New languages, e.g. UPC:
    - shared pointers
    - upc_memput()
- Code for optimized portion may be proprietary but needs to use publicly available libraries
- Optimizations need to be described but not necessarily in detail – possible use in application tuning
- Attempting to capture: invested effort per flop rate gained
  - Hence the need for baseline submission
- There can be more than one optimized submission for a single base submission (if a given architecture allows for many optimizations)

Base Submission Rules Summary

- Compile and load options
  - Compiler or loader flags which are supported and documented by the supplier are allowed
  - These include porting, optimization, and preprocessor invocation
- Libraries
  - Linking to optimized versions of the following libraries is allowed
    - BLAS
    - MPI
    - FFT
  - Acceptable use of such libraries is subject to the following rules:
    - All libraries used shall be disclosed with the results submission. Each library shall be identified by library name, revision, and source (supplier). Libraries which are not generally available are not permitted unless they are made available by the reporting organization within 6 months.
    - Calls to library subroutines should have equivalent functionality to that in the released benchmark code. Code modifications to accommodate various library call formats are not allowed

Optimized Submission Rules Summary

- Only computational (timed) portion of the code can be changed
  - Verification code can not be changed
- Calculations must be performed in 64-bit precision or the equivalent
  - Codes with limited calculation accuracy are not permitted
- All algorithm modifications must be fully disclosed and are subject to review by the HPC Challenge Committee
  - Passing the verification test is a necessary condition for such an approval
  - The replacement algorithm must be as robust as the baseline algorithm
    - For example — the Strassen Algorithm may not be used for the matrix multiply in the HPL benchmark, as it changes the operation count of the algorithm
- Any modification of the code or input data sets — which utilizes knowledge of the solution or of the verification test — is not permitted
- Any code modification to circumvent the actual computation is not permitted

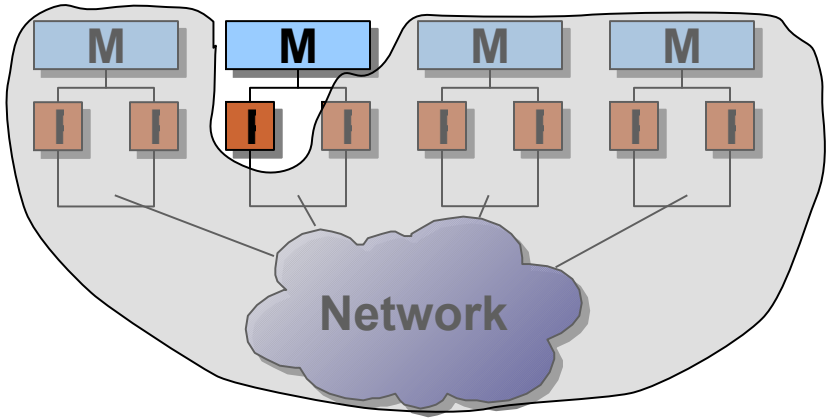
HPCC Tests at a Glance

1. **HPL**
2. **DGEMM**
3. **STREAM**
4. **PTRANS**
5. **RandomAccess**
6. **FFT**
7. **b_eff**

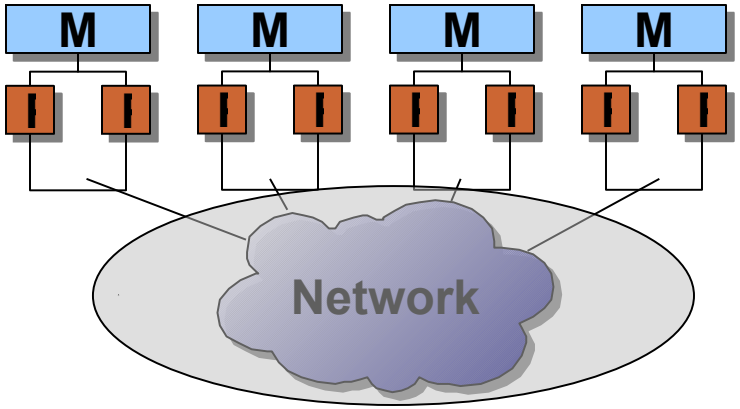
- Scalable framework — Unified Benchmark Framework
  - By design, the HPC Challenge Benchmarks are scalable with the size of data sets being a function of the largest HPL matrix for the tested system

HPCC Testing Scenarios

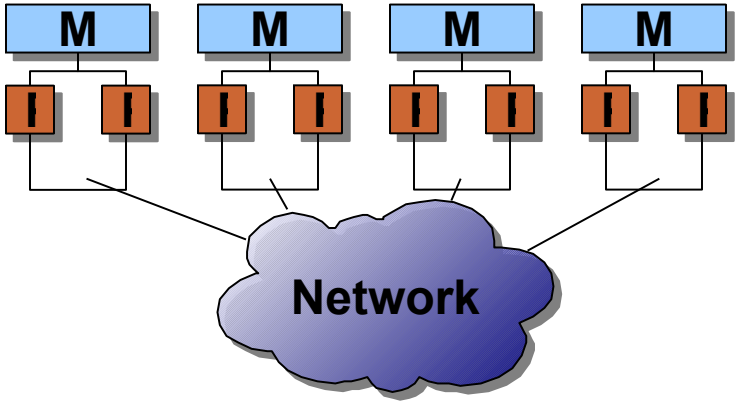
1. Local
    1. Only single process computes



2. Embarrassingly parallel
    1. All processes compute and do not communicate (explicitly)



3. Global
    1. All processes comput and communicate



4. Network only

Naming Conventions

Examples:

2. **G-HPL**
3. **S-STREAM-system**

HPCC Tests - HPL

- **HPL = High Performance Linpack**
- **Objective: solve system of linear equations**

$$Ax=b \qquad A \in \boldsymbol{R}^{n \times n}\, x, b \in \boldsymbol{R}$$

- **Method: LU factorization with partial row pivoting**
- **Performance:** $({}^{2}/_{3} n^3 + {}^{3}/_{2} n^2) / t$
- **Verification: scaled residuals must be small**

$$|| Ax-b || / (\varepsilon\, ||A||\, ||x||\, n)$$

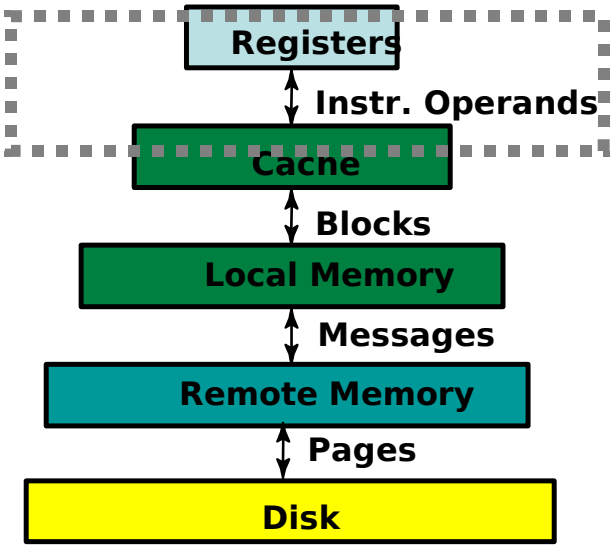
- **Restrictions:**
  - **No complexity reducing matrix-multiply**
    - **(Strassen, Winograd, etc.)**
  - **64-bit precision arithmetic through-out**
    - **(no mixed precision with iterative refinement)**

HPL:TOP500 and HPCC Implementation Comparison

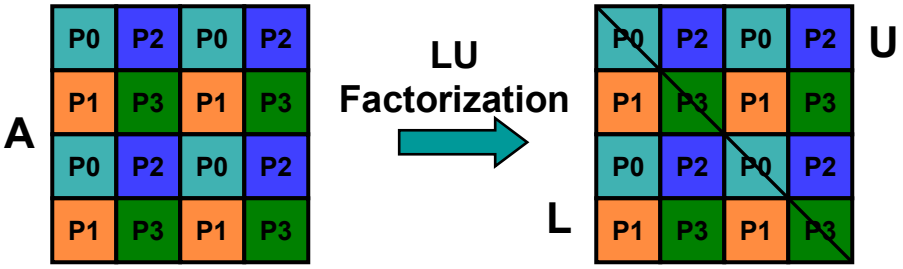
	TOP500	HPCC Base	HPCC Optimized
Disclose code	Not required	Reference code	Describe
Floating-point in 64-bits	Yes	Yes	Yes
Numerics of solution reported	No	Yes	Yes
N-half reported	Yes	No	No
Reporting frequency	Bi-annual	Continuous	Continuous

HPCC HPL: Further Details

- High Performance Linpack (HPL) solves a system Ax = b
- Core operation is a LU factorization of a large MxM matrix
- Results are reported in floating point operations per second (flop/s)



Parallel Algorithm



2D block cyclic distribution is used for load balancing

- Linear system solver (requires all-to-all communication)
- Stresses local matrix multiply performance
- DARPA HPCS goal: 2 Pflop/s (8x over current best)

HPCC Tests - DGEMM

- **DGEMM = Double-precision General Matrix-matrix Multiply**
- **Objective: compute matrix**

$$C \leftarrow \alpha AB + \beta C \qquad A,B,C \in \mathbf{R}^{n \times n} \qquad \alpha,\beta \in \mathbf{R}$$

- **Method: standard multiply (maybe optimized)**
- **Performance:** $2n^3/t$
- **Verification: Scaled residual has to be small**

$$|| x - y || / (\varepsilon\, n\, || y ||)$$

where x and y are vectors resulting from multiplication by a random vector of left and right hand size of the objective expression

- **Restrictions:**
  - **No complexity reducing matrix-multiply**
    - **(Strassen, Winograd, etc.)**
  - **Use only 64-bit precision arithmetic**

HPCC Tests - STREAM

- **STREAM is a test that measures sustainable memory bandwidth (in Gbyte/s) and the corresponding computation rate for four simple vector kernels**
- **<u>Objective</u>: set a vector to a combination of other vectors**

  COPY: $c = a$

  SCALE: $b = \alpha c$

  ADD: $c = a + b$

  TRIAD: $a = b + \alpha c$
- **<u>Method</u>: simple loop that preserves the above order of operations**
- **<u>Performance</u>:** $2n/t$ **or** $3n/t$
- **<u>Verification</u>: scalre residual of computed and reference vector needs to be small**

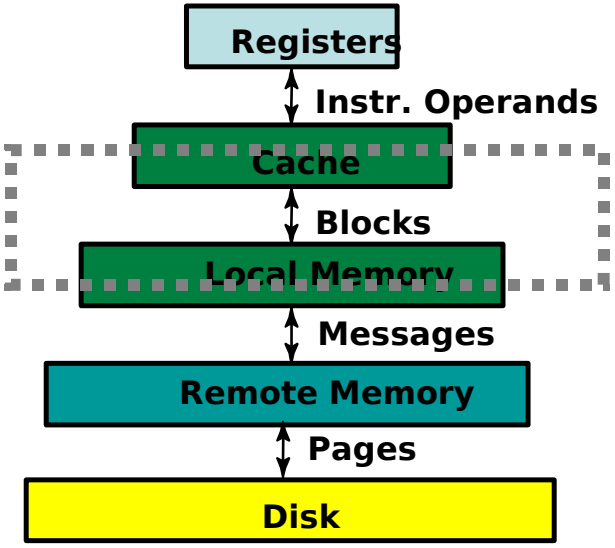
  $|| x - y || / (\varepsilon n || y ||)$
- **<u>Restrictions</u>:**
  - **Use only 64-bit precision arithmetic**

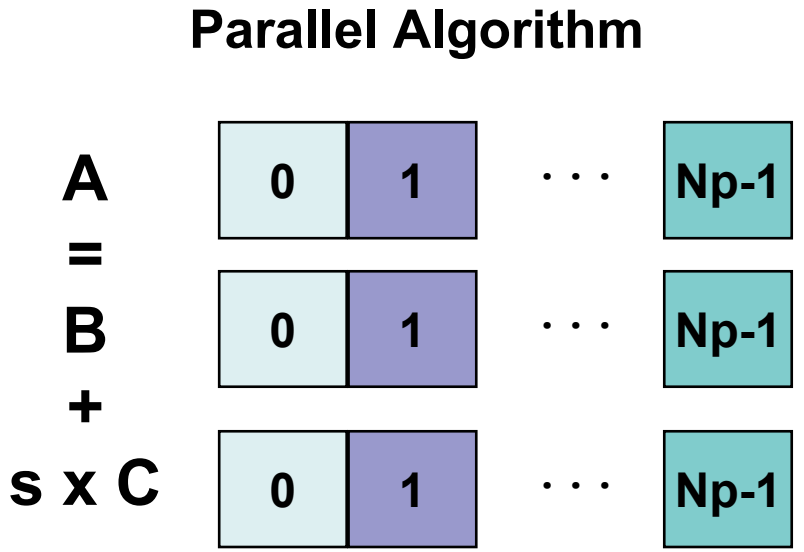
Original and HPCC STREAM Codes

	STREAM	HPCC Base	HPCC Optimized
Programming Language	Fortran (or C)	C only	Any
Data alignment	Implicit	OS-specific	OS-specific
Vectors' size	Exceeds caches, Compile time	Whole memory, Runtime	Whole memory, Runtime
Reporting Multiple of Number of Processors	No	Possible	Possible

HPCC STREAM: Further Details

- Performs scalar multiply and add
- Results are reported in bytes/second





- Basic operations on large vectors (requires no communication)
- Stresses local processor to memory bandwidth
- DARPA HPCS goal: 6.5 Pbyte/s (40x over current best)

HPCC Tests - PTRANS

- **PTRANS = Parallel TRANSpose**
- **Objective: update matrix with sum of its transpose and another matrix**

$$A = A^T + B \qquad A, B \in \boldsymbol{R}^{n \times n}$$

- **Method: standard distributed memory algorithm**
- **Performance:** n^2/t
- **Verification: scaled residual between computed and reference matrix needs to be small**

$$|| A_0 - A || / (\varepsilon \, n \, || A_0 ||)$$

- **Restrictions:**
  - **Use only 64-bit precision arithmetic**
  - **The same data distribution method as HPL**

HPCC PTRANS: Further Details

1	2	3	1	2	3	1	2	3
4	5	6	4	5	6	4	5	6
7	8	9	7	8	9	7	8	9
1	2	3	1	2	3	1	2	3
4	5	6	4	5	6	4	5	6
7	8	9	7	8	9	7	8	9
1	2	3	1	2	3	1	2	3
4	5	6	4	5	6	4	5	6
7	8	9	7	8	9	7	8	9

- **matrix: 9x9**
- **3x3 process grid**
- **Communicating pairs:**
  - **2-4, 3-7, 6-8**

1	2	3	4	5	6	7	8	9
1	2	3	4	5	6	7	8	9
1	2	3	4	5	6	7	8	9
1	2	3	4	5	6	7	8	9
1	2	3	4	5	6	7	8	9
1	2	3	4	5	6	7	8	9
1	2	3	4	5	6	7	8	9
1	2	3	4	5	6	7	8	9
1	2	3	4	5	6	7	8	9

- **matrix: 9x9**
- **1x9 process grid**
- **Communicating pairs:**
  - **1-2, 1-3, 1-4, 1-5, 1-6, 1-7, 1-8, 1-9, 2-3, …, 8-9**
  - **36 pairs!**

HPCC Tests - RandomAccess

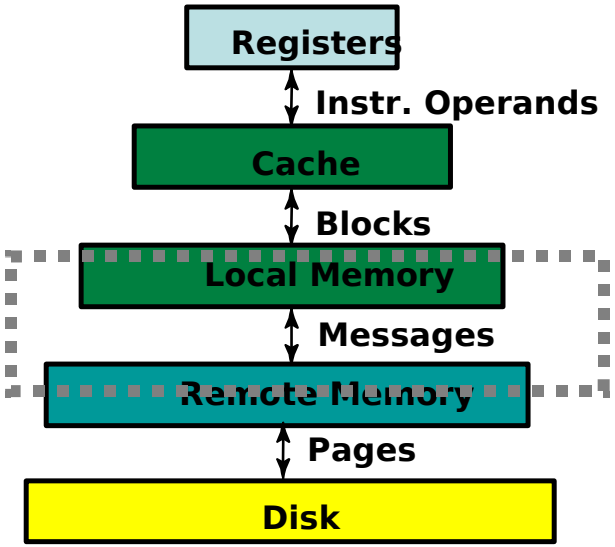
- **RandomAccess calculates a series of integer updates to random locations in memory**
- **Objective: perform computation on Table**

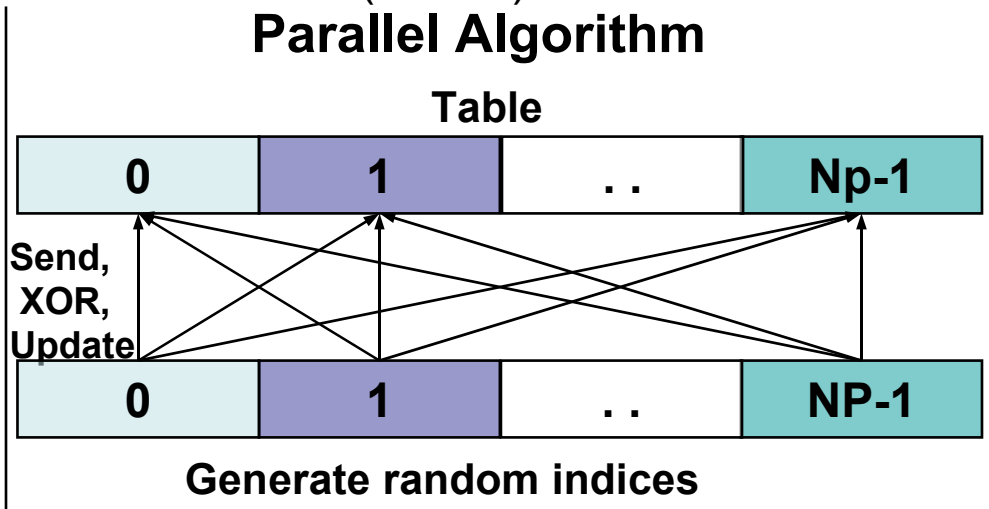
```
Ran = 1;
for (i=0; i<4*N; ++i) {
  Ran= (Ran<<1) ^ (((int64_t)Ran < 0) ? 7:0);
  Table[Ran & (N-1)] ^= Ran;
}
```

- **Method: loop iterations may be independent**
- **Performance:** $4N/t$
- **Verification: up to 1% of updates can be incorrect**
- **Restrictions:**
  - **Use at least 64-bit integers**
  - **About half of memory used for 'Table'**
  - **Parallel look-ahead limited to 1024 (limit locality)**

HPCC RandomAccess: Further Details

- Randomly updates N element table of unsigned integers
- Each processor generates indices, sends to all other processors, performs XOR
- Results are reported in Giga Updates Per Second (GUPS)





- Randomly updates memory (requires all-to-all communication)
- Stresses interprocessor communication of *small* messages
- DARPA HPCS goal: 64,000 GUPS (2000x over current best)

HPCC Tests - FFT

- **FFT = Fast Fourier Transform**
- **Objective: compute discrete Fourier Transform**

  $z_k = \Sigma x_j$ **exp**$(-2\pi\sqrt{-1}\, jk/n)$ $\qquad x,z \in \mathbf{C}^n$

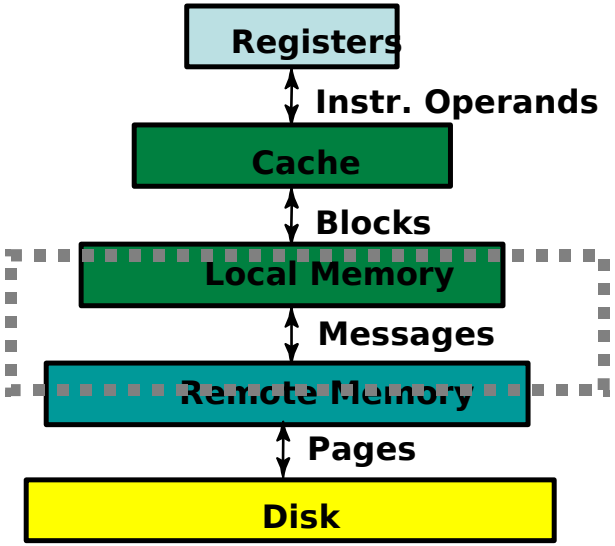
- **Method: any standard framework (maybe optimized)**
- **Performance:** $5n\log_2 n/t$
- **Verification: scaled residual for inverse transform of computed vector needs to be small**

  $|| x - x^{(0)} || / (\varepsilon \log_2 n)$

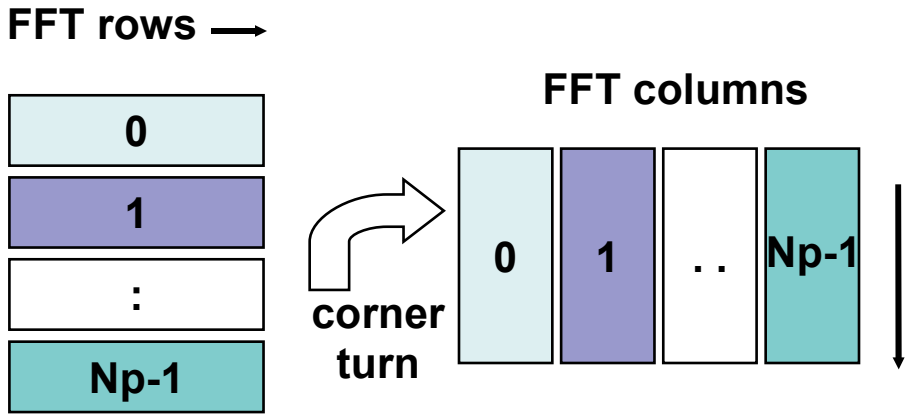
- **Restrictions:**
  - **Use only 64-bit precision arithmetic**
  - **Result needs to be in-order (not bit-reversed)**

HPCC FFT: Further Details

- 1D Fast Fourier Transforms an N element complex vector
- Typically done as a parallel 2D FFT
- Results are reported in floating point operations per second (flop/s)



Parallel Algorithm



- FFT a large complex vector (requires all-to-all communication)
- Stresses interprocessor communication of *large* messages
- DARPA HPCS goal: 0.5 Pflop/s (200x over current best)

HPCC Tests – b_eff

- b_eff measures effective bandwidth and latency of the interconnect
- Objective: exchange 8 (for latency) and 2000000 (for bandwidth) messages in ping-pong, natural and random ring patterns
- Method: use standard MPI point-to-point routines
- Performance: n/t (for bandwidth)
- Verification: simple checksum on received bits
- Restrictions:
  - The messaging routines have to conform to the MPI standard

HPCC Awards Overview

- Goals
  - Increase awareness of HPCC
  - Increase awareness of HPCS and its goals
  - Increase number of HPCC submissions
    - Expanded view of largest supercomputing installations
- Means
  - HPCwire sponsorships and press coverage
  - HPCS mission partners' contribution
  - HPCS vendors' contribution

HPCC Awards Rules

- Class 1: Best Performance
  - Figure of merit: raw system performance
  - Submission must be valid HPCC database entry
    - Side effect: populate HPCC database
  - 4 categories: HPCC components
    - HPL
    - STREAM-system
    - RandomAccess
    - FFT
  - Award certificates
    - 4x $500 from HPCwire

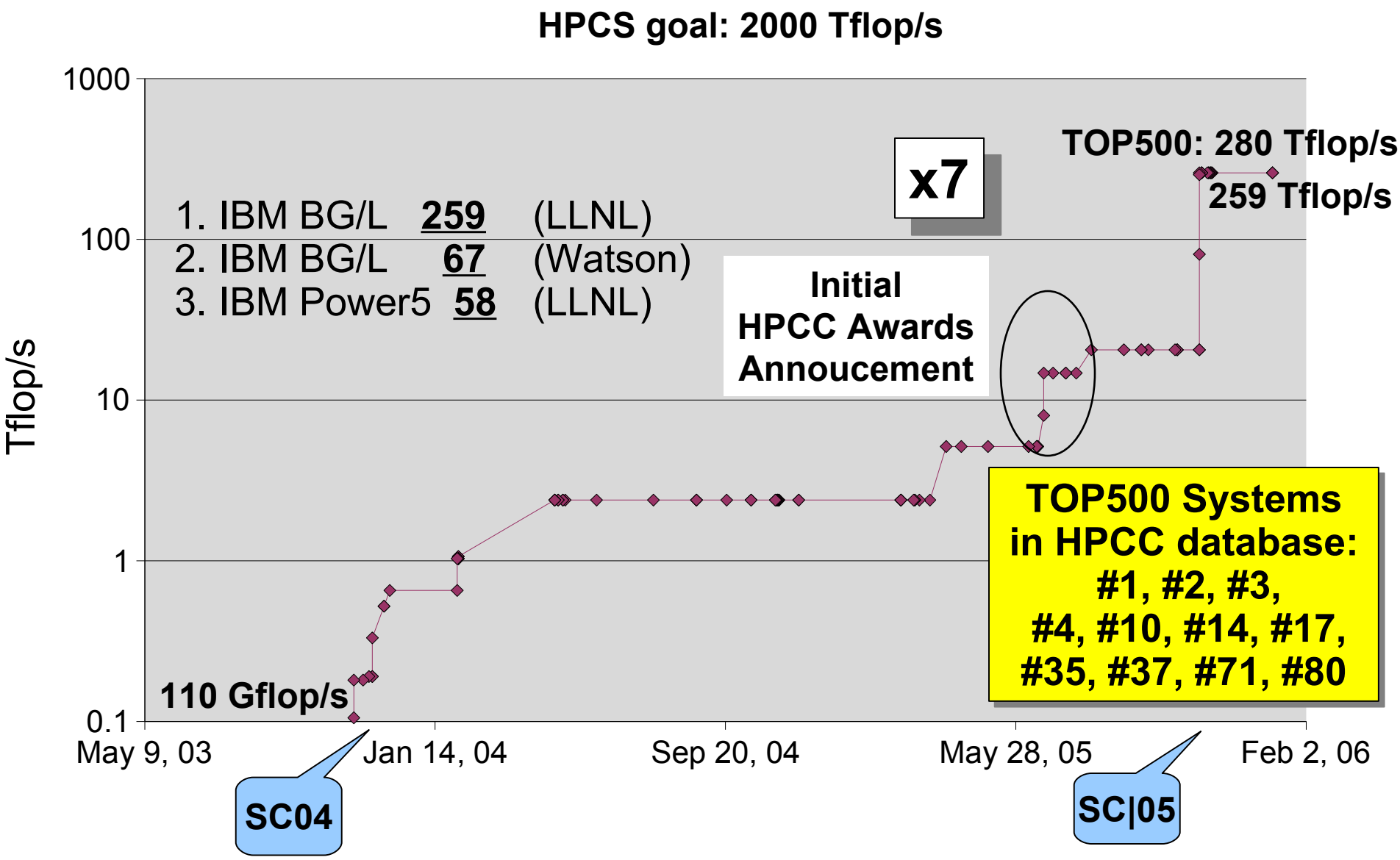
- Class 2: Most Productivity
  - Figure of merit: performance (50%) and elegance (50%)
    - Highly subjective
    - Based on committee vote
  - Submission must implement at least 3 out of 4 Class 1 tests
    - The more tests the better
  - Performance numbers are a plus
  - The submission process:
    - Source code
    - "Marketing brochure"
    - SC06 BOF presentation
  - Award certificate
    - $1500 from HPCwire

HPCwire

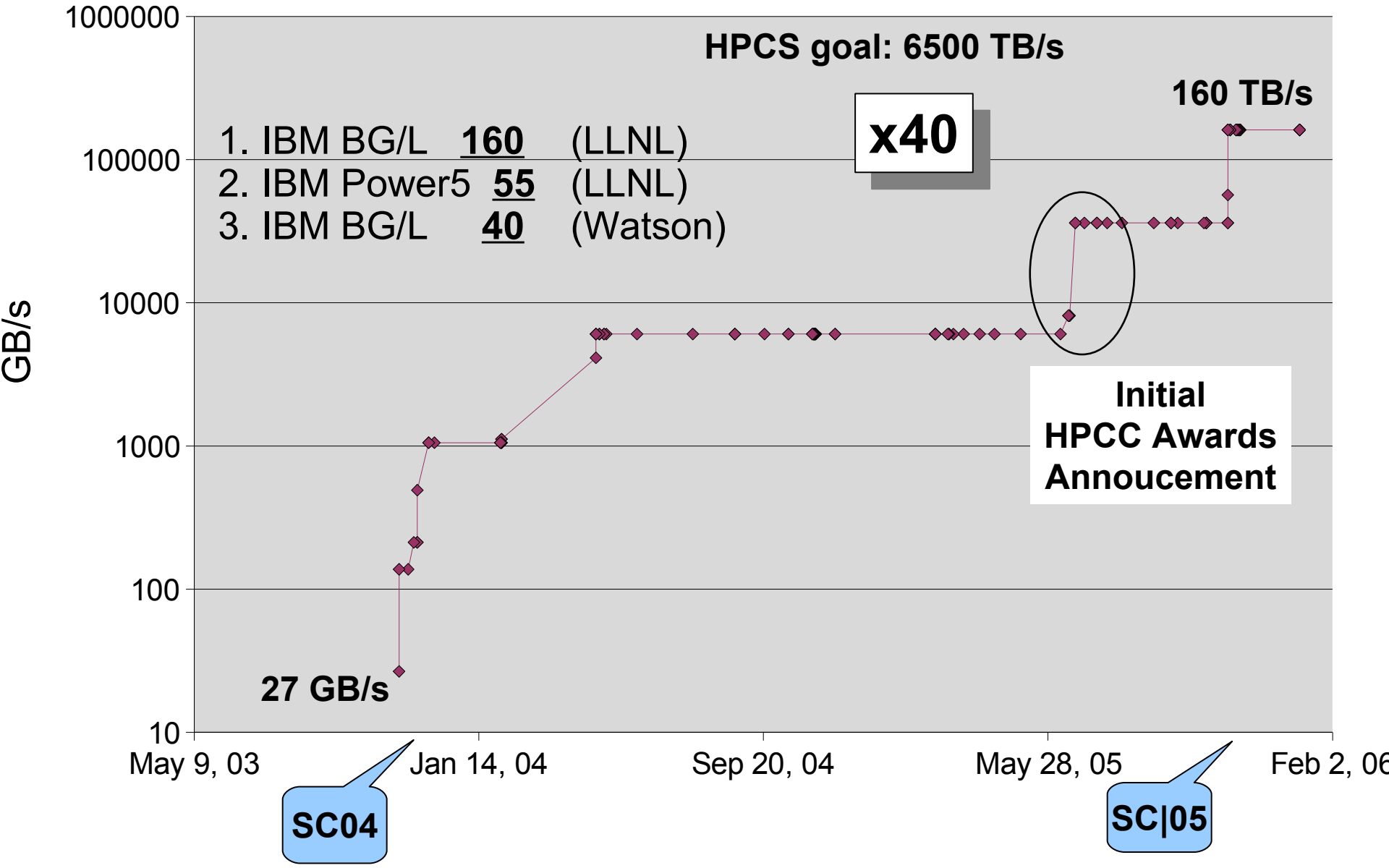
HPCC Awards Committee

- **David Bailey**
  LBNL NERSC
- **Jack Dongarra (Co-Chair)**
  Univ. of Tenn/ORNL
- **Jeremy Kepner (Co-Chair)**
  MIT Lincoln Lab
- **Bob Lucas**
  USC/ISI
- **Rusty Lusk**
  Argonne National Lab
- **Piotr Luszczek**
  Univ. of Tennessee
- **John McCalpin**
  AMD
- **Rolf Rabenseifner**
  HLRS Stuttgart
- **Daisuke Takahashi**
  Univ. of Tsukuba

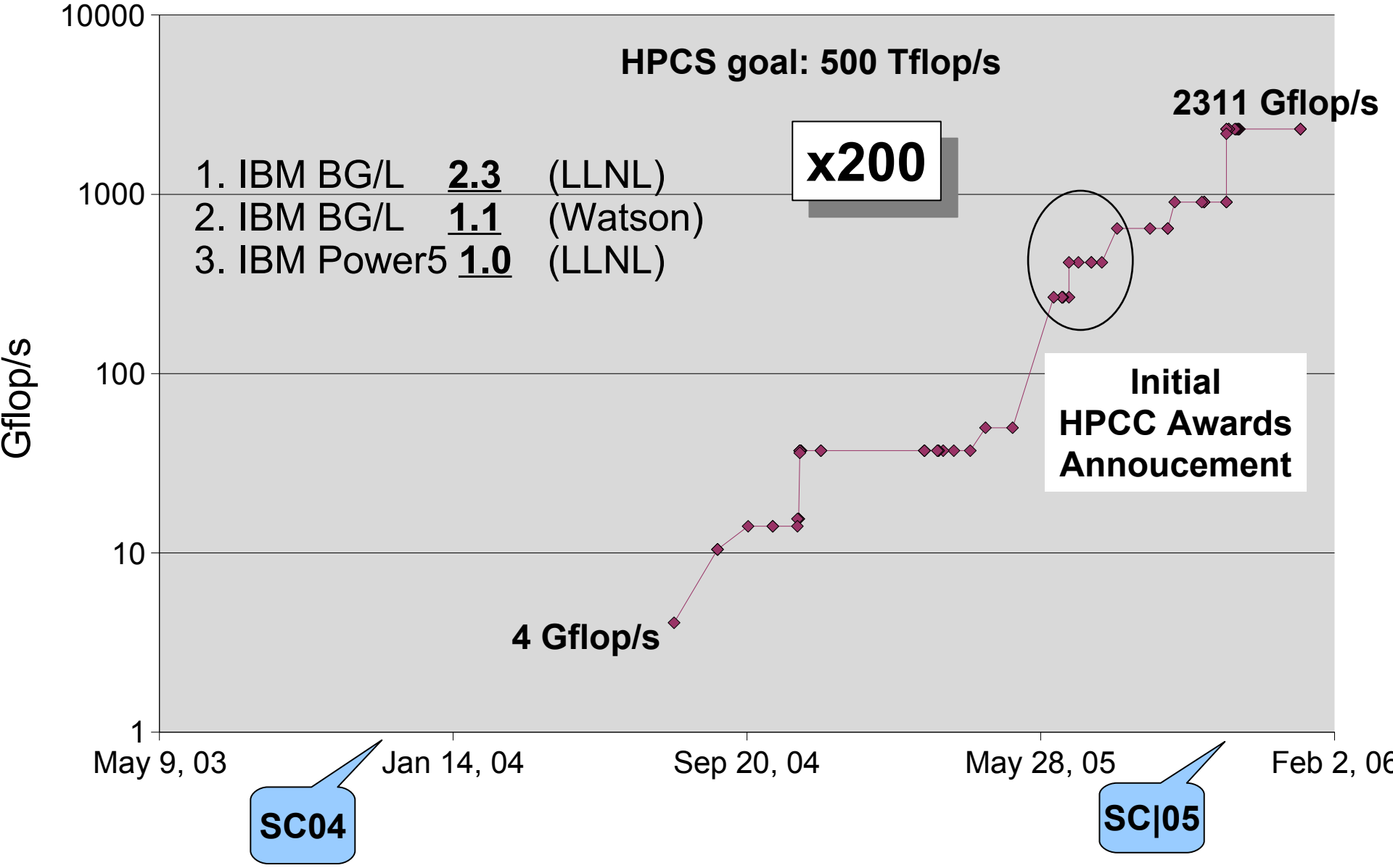
SC|05 HPCC Awards Class 1 - HPL



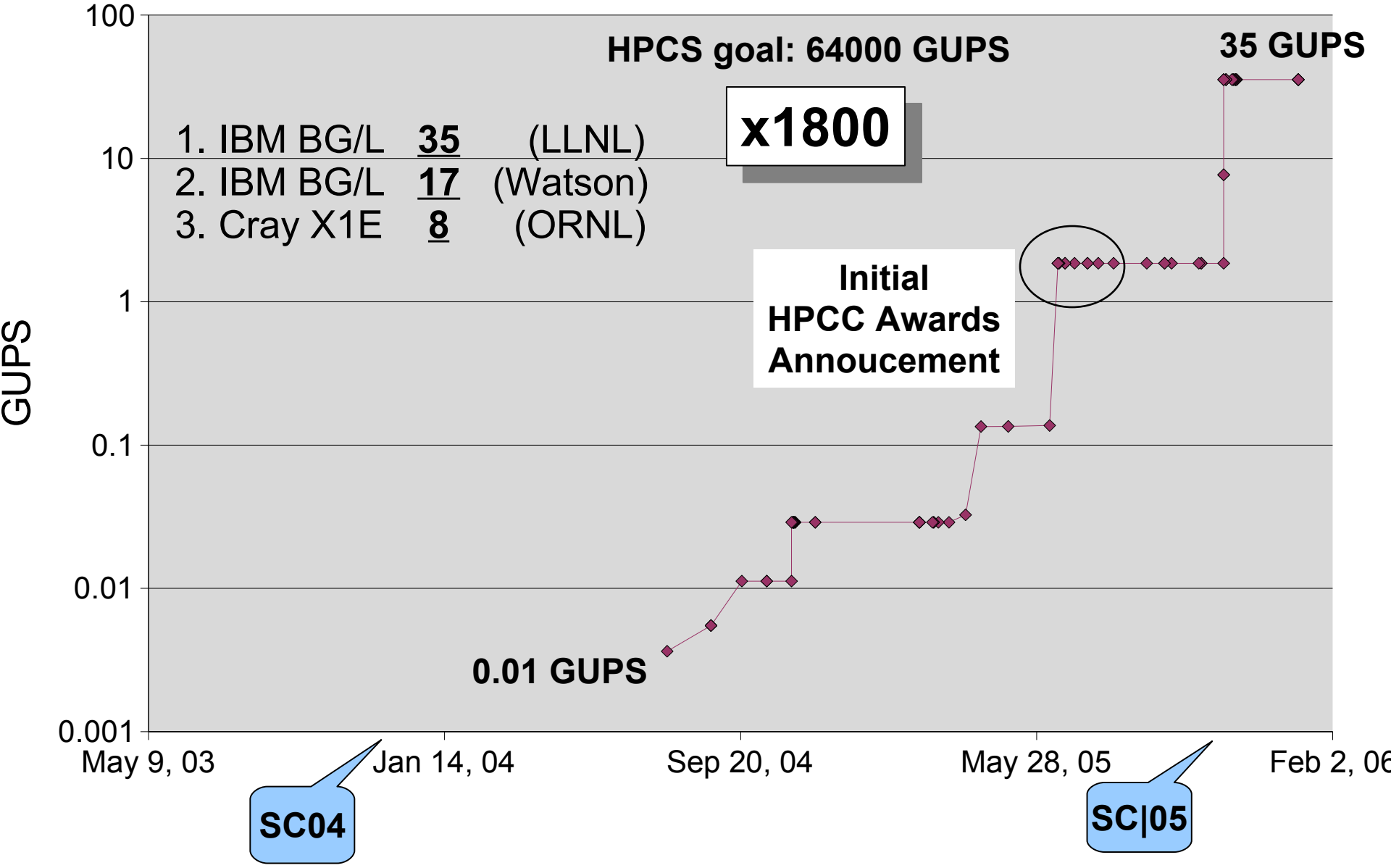
SC|05 HPCC Awards Class 1 – STREAM-sys



SC|05 HPCC Awards Class 1 – FFT



SC|05 HPCC Awards Class 1 –RandomAccess



SC|05 HPCC Awards Class 2

Language	HPL	RandomAccess	STREAM	FFT
Python+MPI		√	√	
pMatlab	√	√	√	√
Cray MTA C		√		√
UPCx3	√	√	√	
Cilk	√	√	√	√
Parallel Matlab	√	√	√	√
MPT C	√			√
OpenMP, C++		√	√	
StarP	√		√	
HPF	√			√

Sample submission from committee members

Winners

Finalists

SC06 HPCC Awards BOF

Awards will be presented at the SC06 HPC Challenge BOF

Tuesday, November 14, 2006

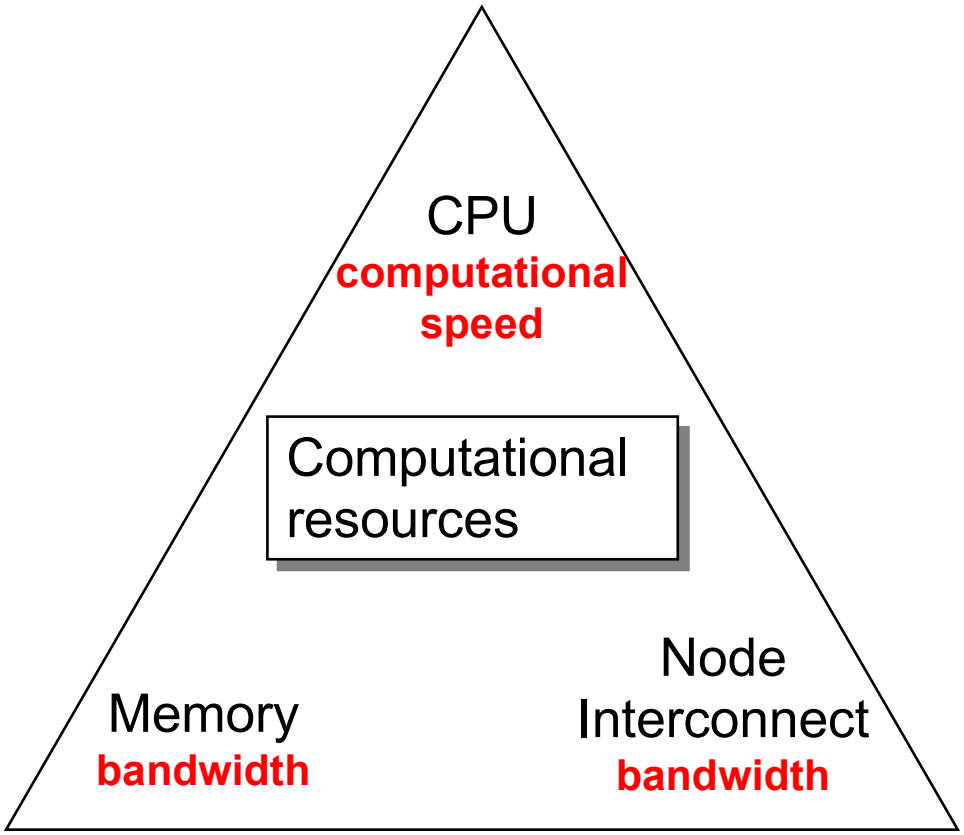
In Ballroom B-C

12:15-1:15

HPCC b_eff Analysis Outline

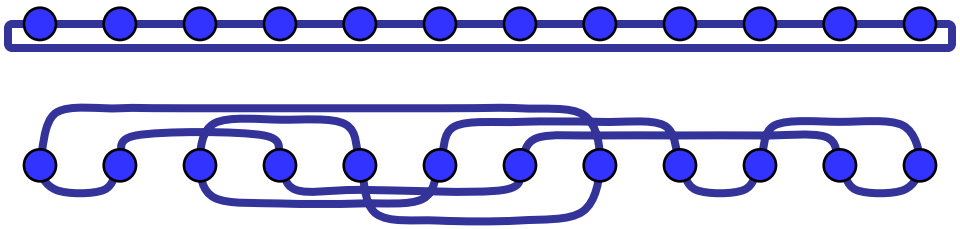
- How HPC Challenge Benchmark (HPCC) data can be used to analyze the balance of HPC systems
  - Details on ring based benchmarks
- Resource based ratios
  - Inter-node bandwidth and
  - memory bandwidth
  - versus computational speed
  - Comparison mainly based on public HPCC data

HPCC and Computational Resources



Random and Natural Ring B/W and Latency

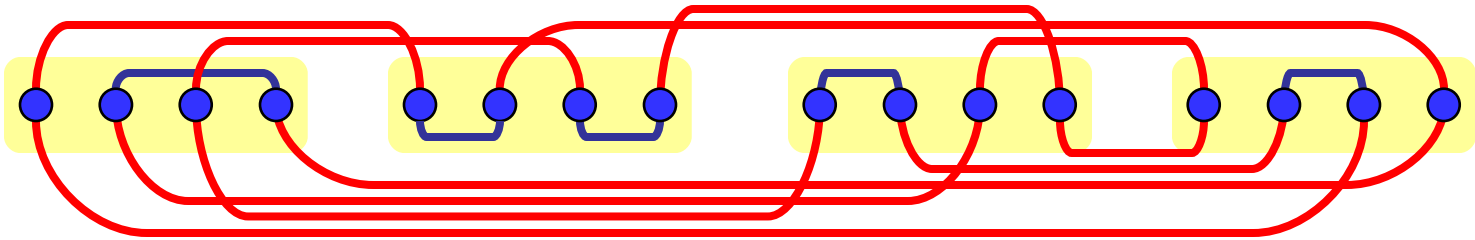
- **Parallel communication pattern on all MPI processes ( )**
  - **Natural ring**
  - **Random ring**



- **Bandwidth per process**
  - **Accumulated message size / wall-clock time / number of processes**
  - **On each connection messages in both directions**
  - **With 2xMPI_Sendrecv and MPI non-blocking → best result is used**
  - **Message size = 2,000,000 bytes**
- **Latency**
  - **Same patterns, message size = 8 bytes**
  - **Wall-clock time / (number of sendrecv per process)**

Inter-node B/W on Clusters of SMP Nodes

- **Random Ring**
  - **Reflects the other dimension of a Cartesian domain decomposition and**
  - **Communication patterns in unstructured grids**
  - **Some connections are inside of the nodes**
  - **Most connections are inter-node**
  - **Depends on #nodes and #MPI processes per node**

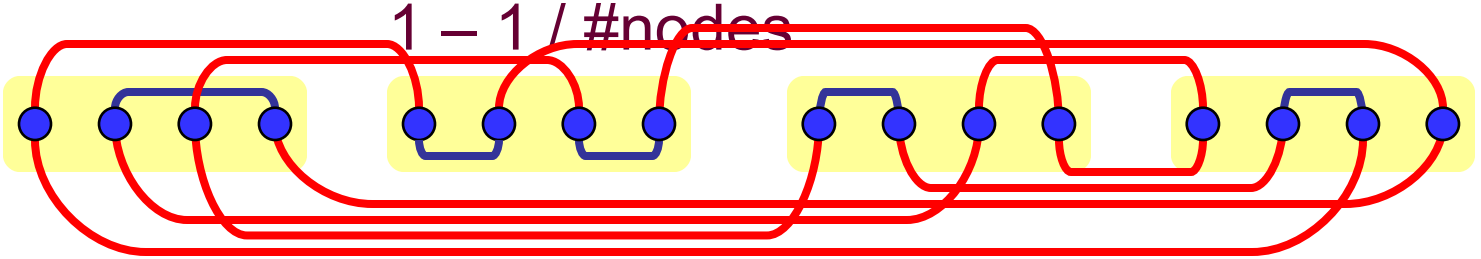


Accumulated Inter-node B/W on Clusters of SMPs

- Accumulated bandwidth

  := bandwidth per process x #processes

  $\sim= \frac{\text{accumulated inter-node bandwidth}}{1 - 1 / \#\text{nodes}}$

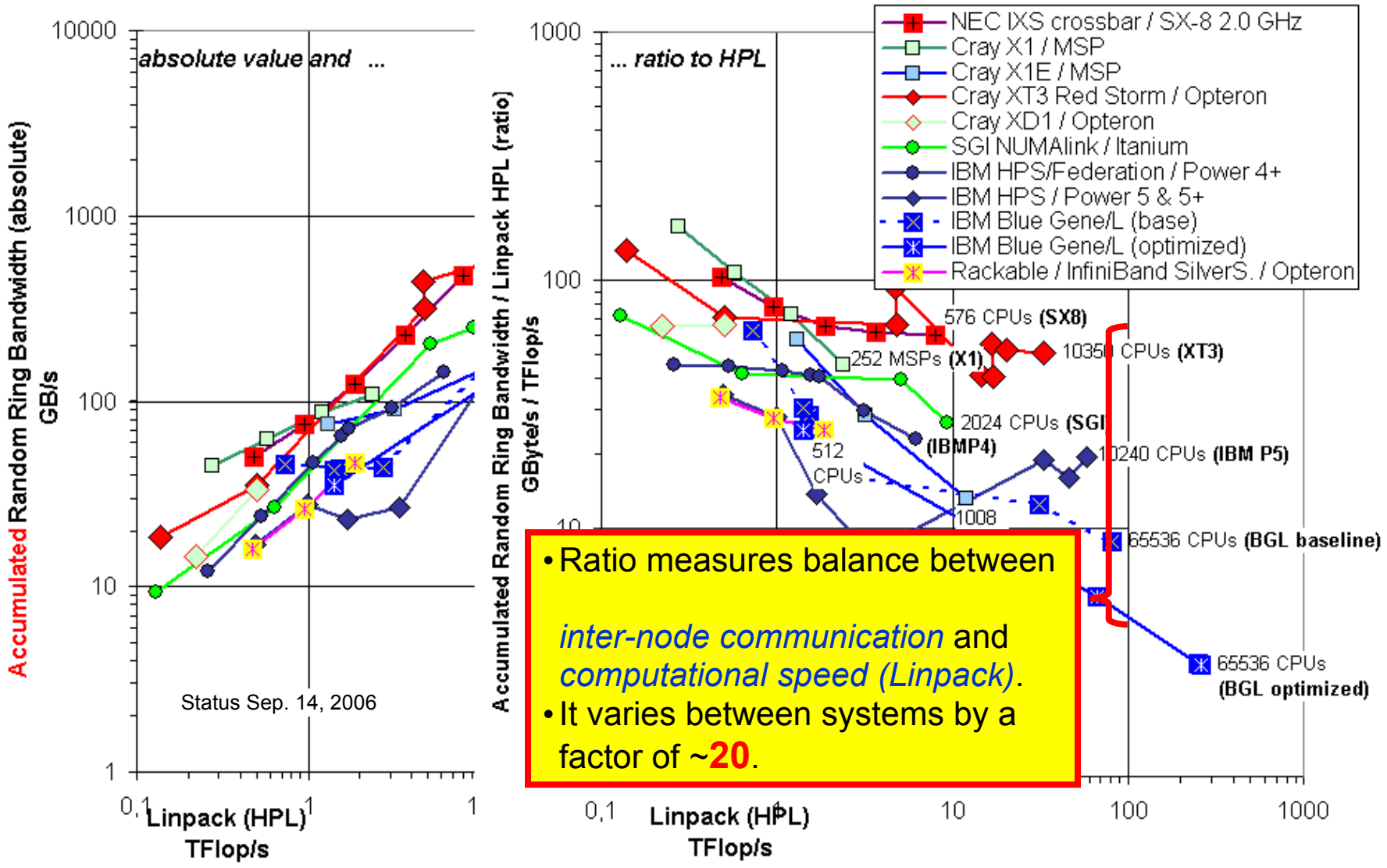


similar to bi-section bandwidth

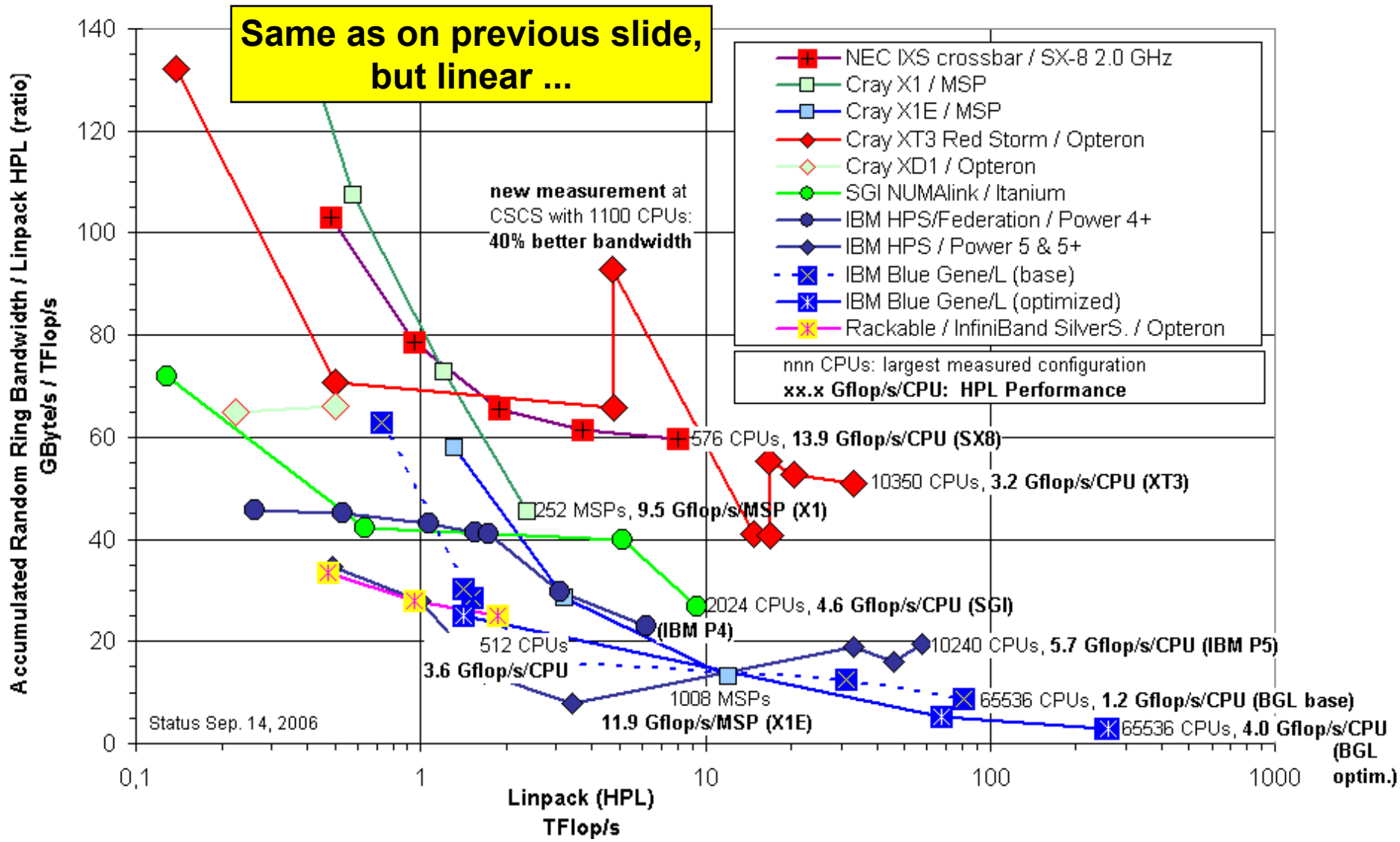
Balance Analysis with HPCC Data

- **Balance can be expressed as a set of ratios**
  - **e.g., accumulated memory bandwidth / accumulated Tflop/s rate**
- **Basis**
  - Linpack (HPL) → Computational Speed
  - Random Ring Bandwidth → Inter-node communication
  - Parallel STREAM Copy or Triad → Memory bandwidth
- **Be careful:**
  - **Some data are presented for the total system**
  - **Some per MPI process (HPL processes), e.g., ring bandwidth**
  - **i.e., balance calculation always with accumulated data on the total system**

Balance: Random Ring B/W and HPL

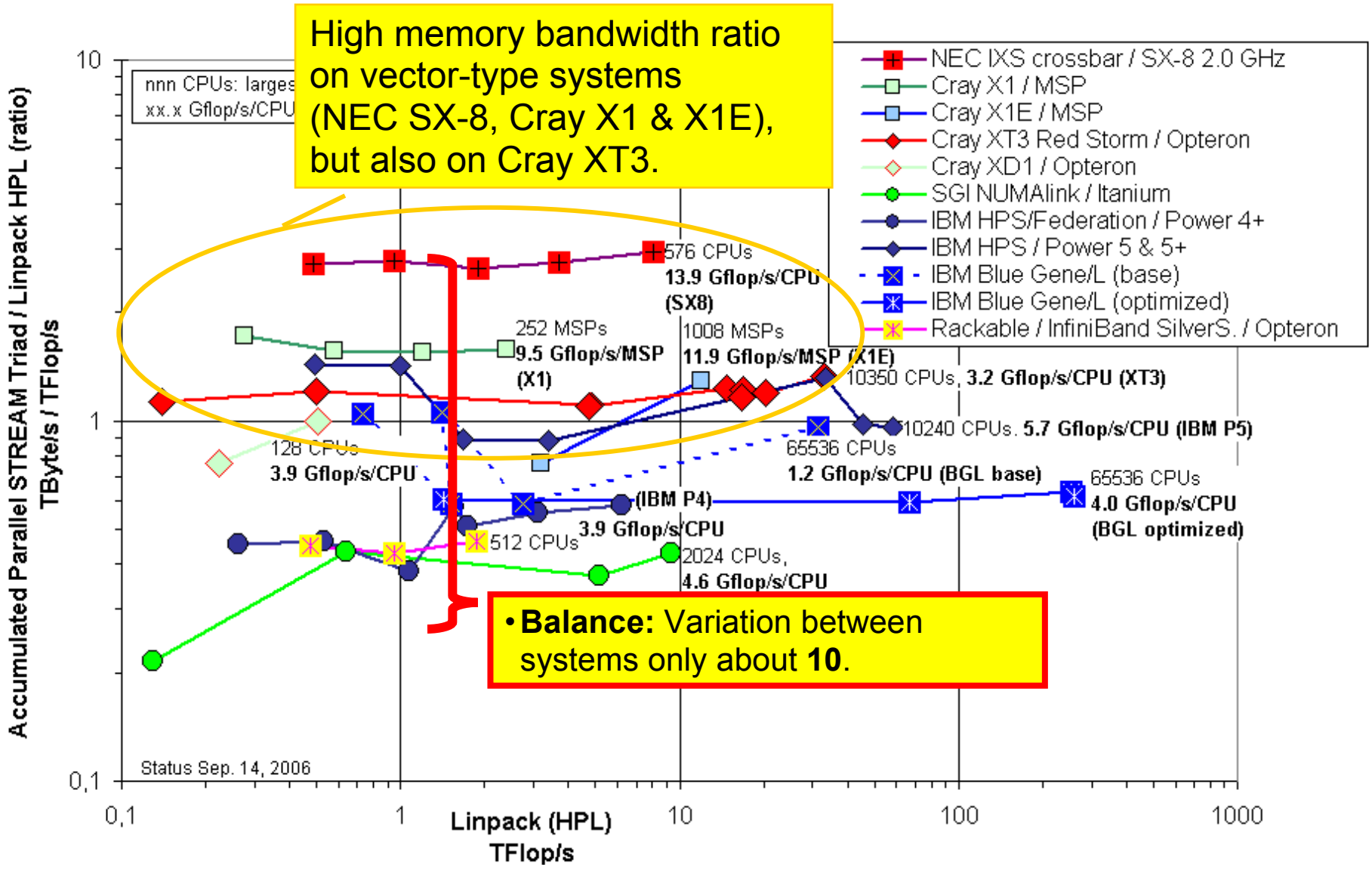


Balance: Random Ring B/W and CPU Speed

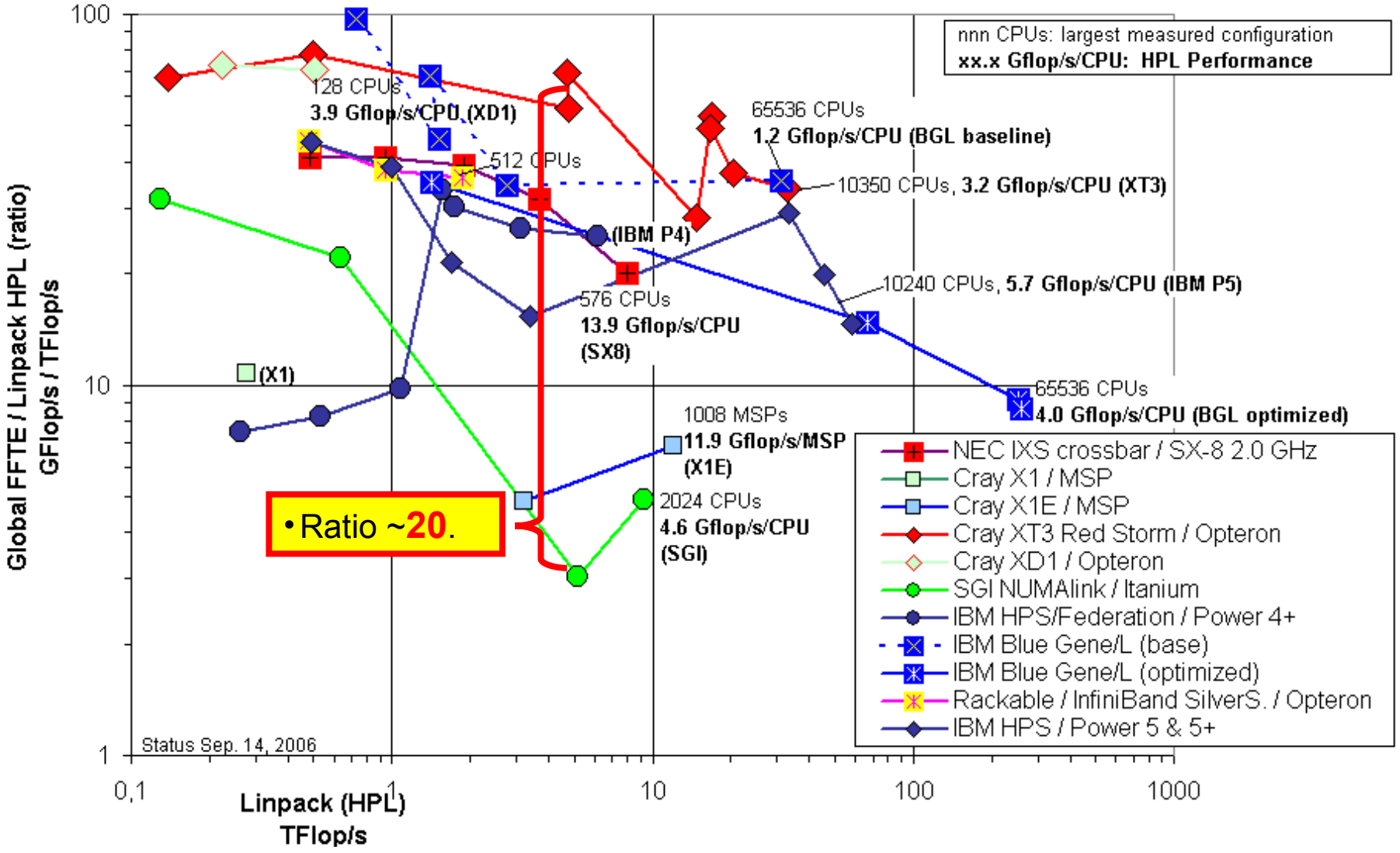


hpcc_2006-09-14_analysis.ppt → Als Webseite speichern → hpcc_2006-09-14_analysis-Diagramme/image006.gif

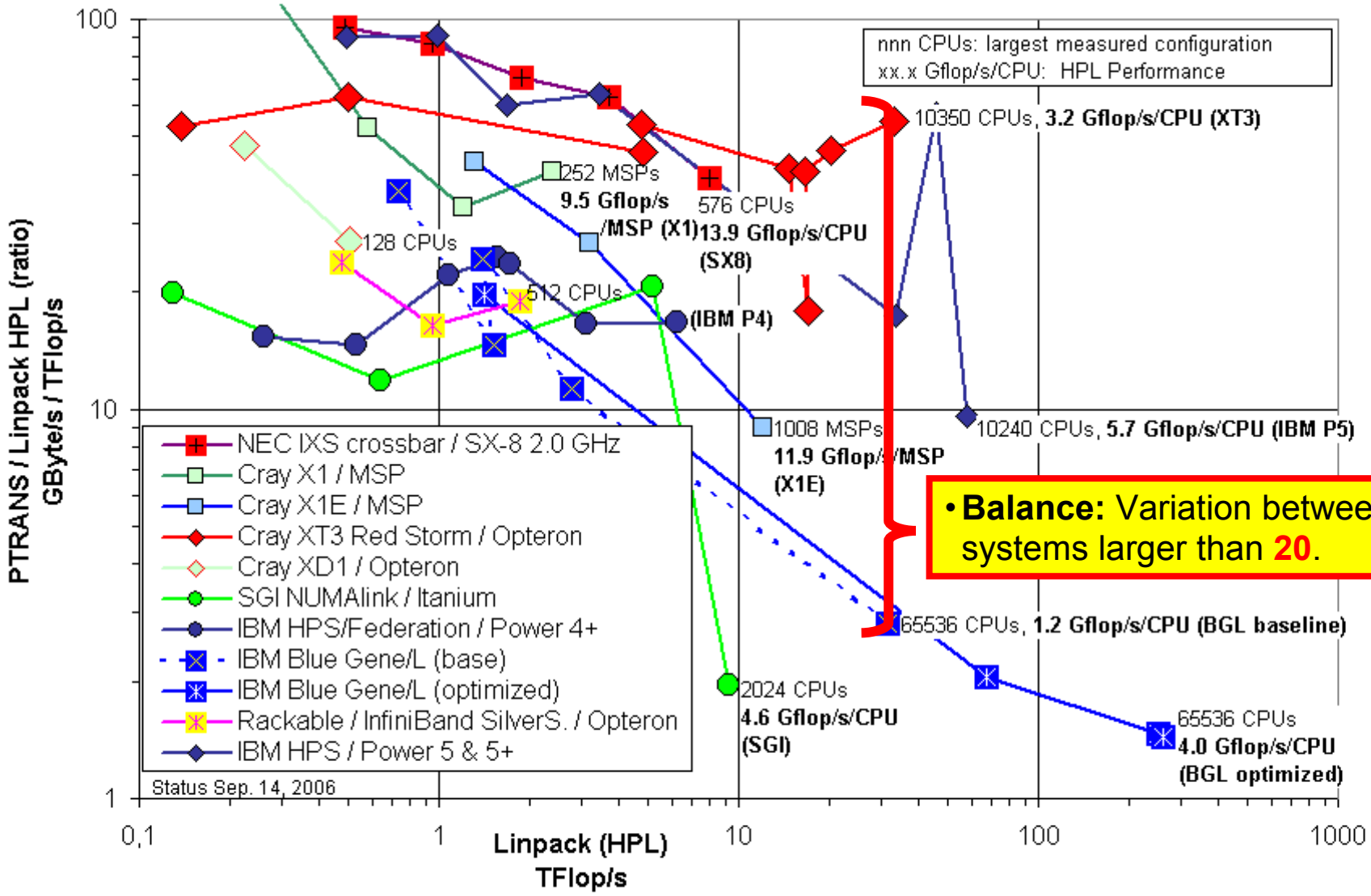
Balance: Memory and CPU Speed



Balance: FFT and CPU



Balance: PTRANS and CPU



- **Balance:** Variation between systems larger than **20**.

hpcc_2006-09-14_analysis.ppt → Als Webseite speichern → hpcc_2006-09-14_analysis-Diagramme/image013.gif

Acknowledgments

- **Thanks to**
  - **all persons and institutions that have uploaded HPCC results.**
  - **Jack Dongarra and Piotr Luszczek for inviting me into the HPCC development team.**
  - **Matthias Müller, Sunil Tiyyagura and Holger Berger for benchmarking on the SX-8 and SX-6 and discussions on HPCC.**
  - **Nathan Wichmann from Cray for Cray XT3 and X1E data.**

- **References**

  - **S. Saini, R. Ciotti,B. Gunney, Th. Spelce, A. Koniges, D. Dossa, P. Adamidis, R. Rabenseifner, S. Tiyyagura, M, Müller, and R. Fatoohi:** Performance Evaluation of Supercomputers using HPCC and IMB Benchmarks. **In the proceedings of the IPDPS 2006 Conference.**
  - **R. Rabenseifner, S. Tiyyagurra, M. Müller:** Network Bandwidth Measurements and Ratio Analysis with the HPC Challenge Benchmark Suite (HPCC). **Proceedings of the 12th European PVM/MPI Users' Group Meeting, EuroPVM /MPI 2005**

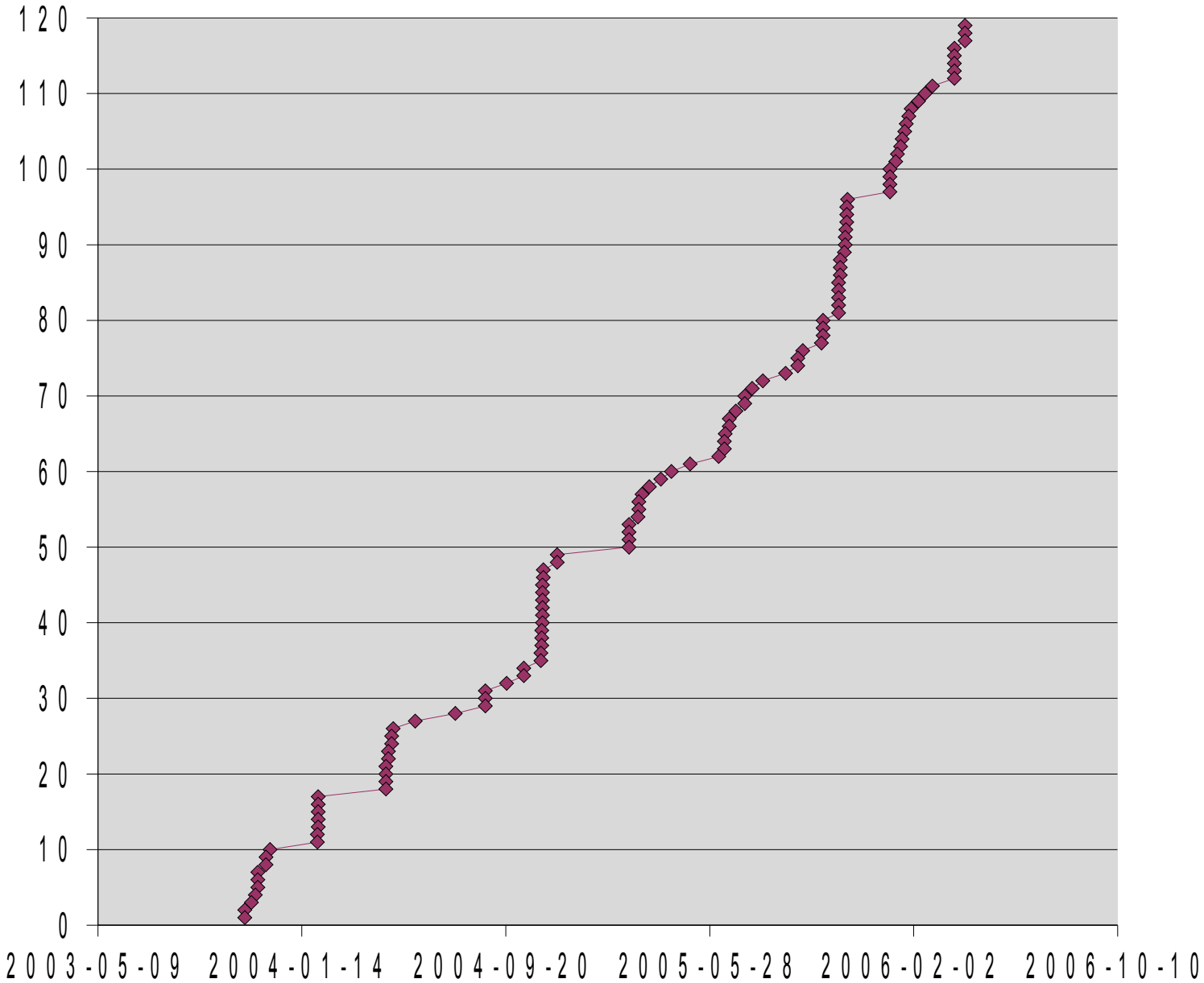
Conclusions

- HPCC is an interesting basis for
  - benchmarking computational resources
  - analyzing the balance of a system
  - scaling with the number of processors
  - with respect to application needs
- HPCC helps to show the strength and weakness of super-computers
- Future super computing should not focus only on Pflop/s in the TOP500
  - Memory and network bandwidth are as same as important to predict real application performance

Reporting Results - Etiquette

- The HPC Challenge Benchmark suite has been designed to permit academic style usage for comparing
  - Technologies
  - Architectures
  - Programming models
- There is an overt attempt to keep HPC Challenge significantly different than "commercialized" benchmark suites
  - Vendors and users can submit results
  - System "cost/price" is not included intentionally
  - No "composite" benchmark metric
- Be cool about comparisons!
- While we can not enforce any rule to limit comparisons observe rules of
  - Academic honesty
  - Good taste

Total Number of HPCC Submissions



Based vs. Optimized Submission

- **Optimized G-RandomAccess is a UPC code**
  - **~125x improvement**

System Information			Run Type	G-HPL TFlop/s	G-PTRANS GB/s	G-Random Access Gup/s	G-FFTE GFlop/s	G-STREAM Triad GB/s	EP STREAM Triad GB/s	EP DGEMM GFlop/s	Random Ring Bandwidth GB/s	Random Ring Latency usec
System - Processor	Speed	Count										
Cray mfeg8 X1E	1.13GHz	248	opt	3.3889	66.01	1.85475	-1	3280.9	13.229	13.564	0.29886	14.58
Cray X1E X1E MSP	1.13GHz	252	base	3.1941	85.204	0.014868	15.54	2440	9.682	14.185	0.36024	14.93

Exploiting Hybrid Programming Model

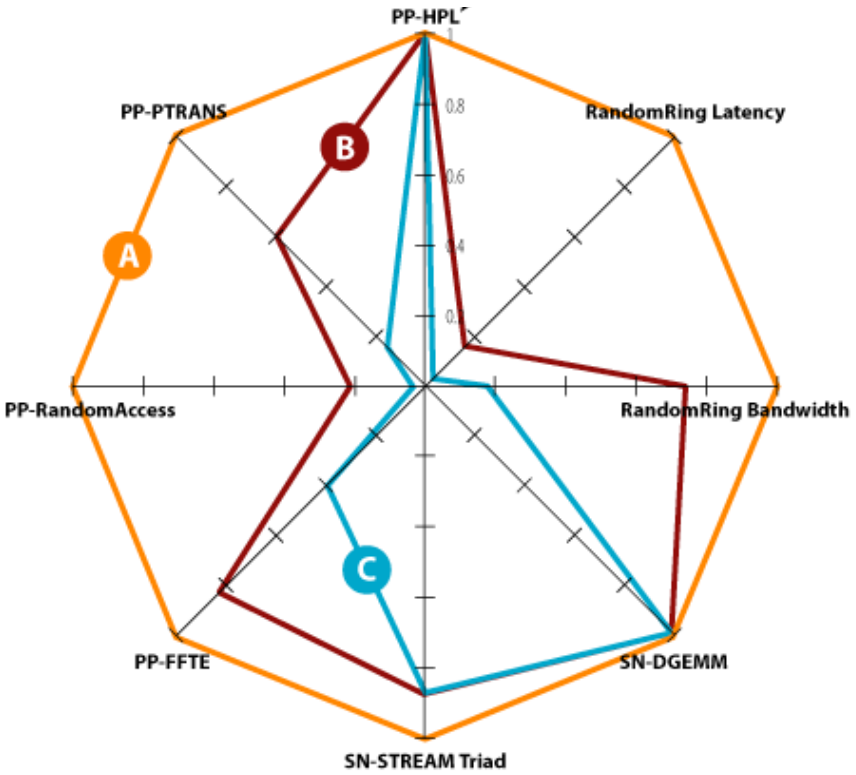
- The NEC SX-7 architecture can permit the definition of threads and processes to significantly enhance performance of the EP versions of the benchmark suite by allocating more powerful “nodes”
  - EP-STREAM
  - EP-DGEMM

System Information					G-HPL	G-PTRANS	G-Random Access	G-FFTE	G-STREAM Triad	EP STREAM Triad	EP DGEMM	Random Ring Bandwidth	Random Ring Latency
System	Speed	Count	Threads	Processes	TFlop/s	GB/s	Gup/s	GFlop/s	GB/s	GB/s	GFlop/s	GB/s	usec
NEC SX-7	0.552GHz	32	16	2	0.2174	16.34	0.000178	1.34	984.3	492.161	140.636	8.14753	4.85
NEC SX-7	0.552GHz	32	1	32	0.2553	20.546	0.000964	11.29	836.9	26.154	8.239	5.03934	14.21

Kiviat Charts: Comparing Interconnects

- AMD Opteron clusters
  - 2.2 GHz
  - 64-processor cluster
- Interconnects
  1. GigE
  2. Commodity
  3. Vendor
- Cannot be differentiated based on:
  - HPL
  - Matrix-matrix multiply
- Available on HPCC website

Kiviat chart (radar plot)



Augmenting TOP500's 26th Edition with HPCC

	Computer	Rmax	HPL	PTRANS	STREAM	FFT	GUPS	Latency	B/W
1	BlueGene/L	281	259	374	160	2311	35.5	6	0.2
2	BGW	91	84	172	50	84	21.6	5	0.2
3	ASC Purple	63	58	576	44	967	0.2	5	3.2
4	Columbia	52	47	91	21	230	0.2	4	1.4
5	Thunderbird	38							
6	Red Storm	36	33	1813	44	1118	1.0	8	1.2
7	Earth Simulator	36							
8	MareNostrum	28							
9	Stella	27							
10	Jaguar	21	20	944	29	855	0.7	7	1.2

Augmenting TOP500's 27th Edition with HPCC

	Computer	Rmax	HPL	PTRANS	STREAM	FFT	GUPS	Latency	B/W
1	BlueGene/L	280.6	259.2	4665.9	160	2311	35.47	5.92	0.159
2	BGW (*)	91	83.9	171.55	50	1235	21.61	4.70	0.159
3	ASC Purple	75.8	57.9	553	55	842	1.03	5.1	3.184
4	Columbia (*)	51.87	46.78	91.31	20	229	0.25	4.23	0.896
5	Tera-10	42.9							
6	Thunderbird	38.27							
7	Fire x4600	38.18							
8	BlueGene eServer	37.33							
9	Red Storm	36.19	32.99	1813.06	43.58	1118	1.02	7.97	1.149
10	Earth Simulator	35.86							

Normalize with Respect to Peak flop/s

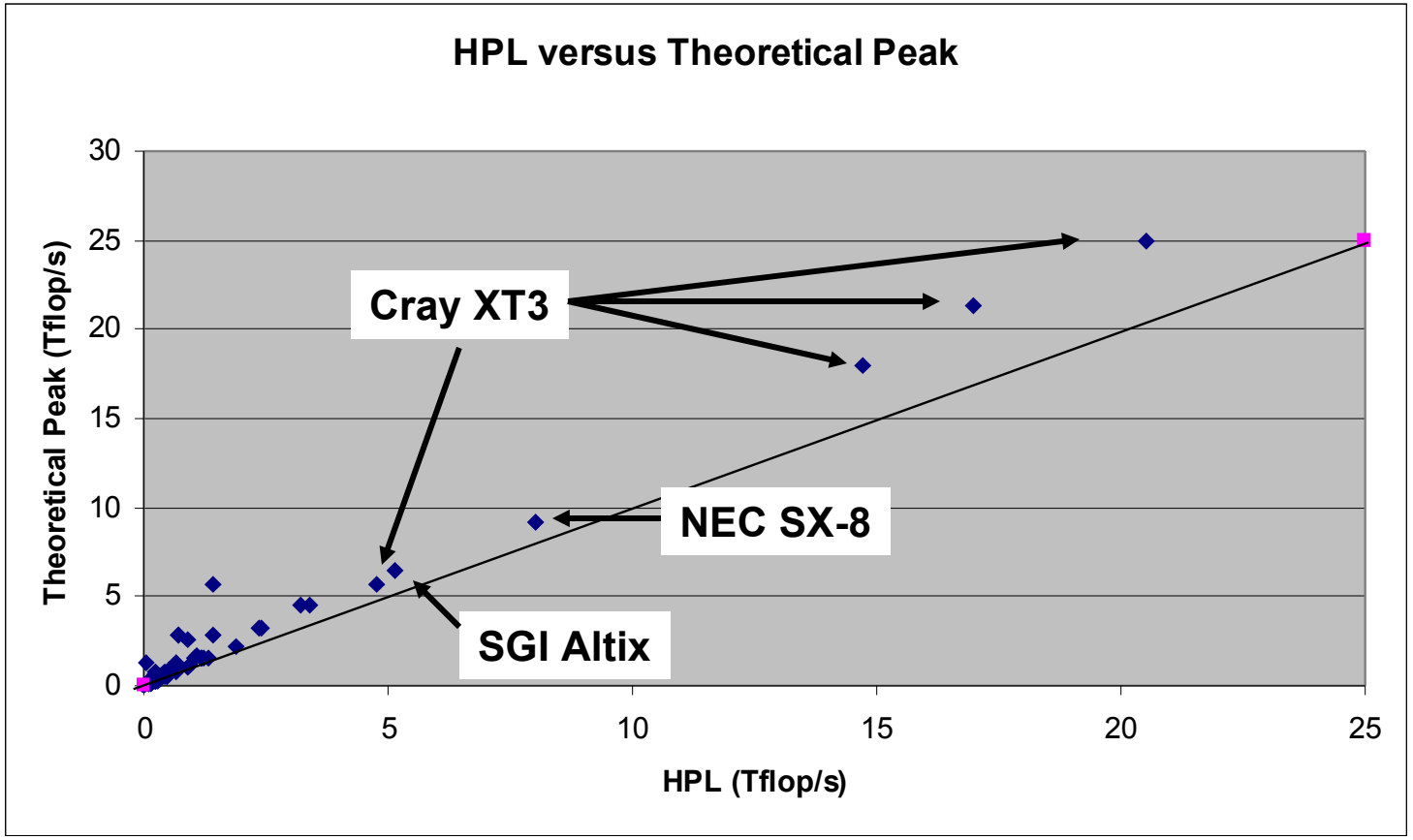
Computer	HPL	RandomAccess	STREAM	FFT
Cray XT3	81.4%	0.031	1168.8	38.3
Cray X1E	67.3%	0.422	696.1	13.4
IBM POWER5	53.5%	0.003	703.5	15.5
IBM BG/L	70.6%	0.089	435.7	6.1
SGI Altix	71.9%	0.003	308.7	3.5
NEC SX-8	86.9%	0.002	2555.9	17.5

Normalizing with peak flop/s cancels out number of processors:

G-HPL / Rpeak = (local-HPL*P) / (local-Rpeak*P) = local-HPL / local-Rpeak

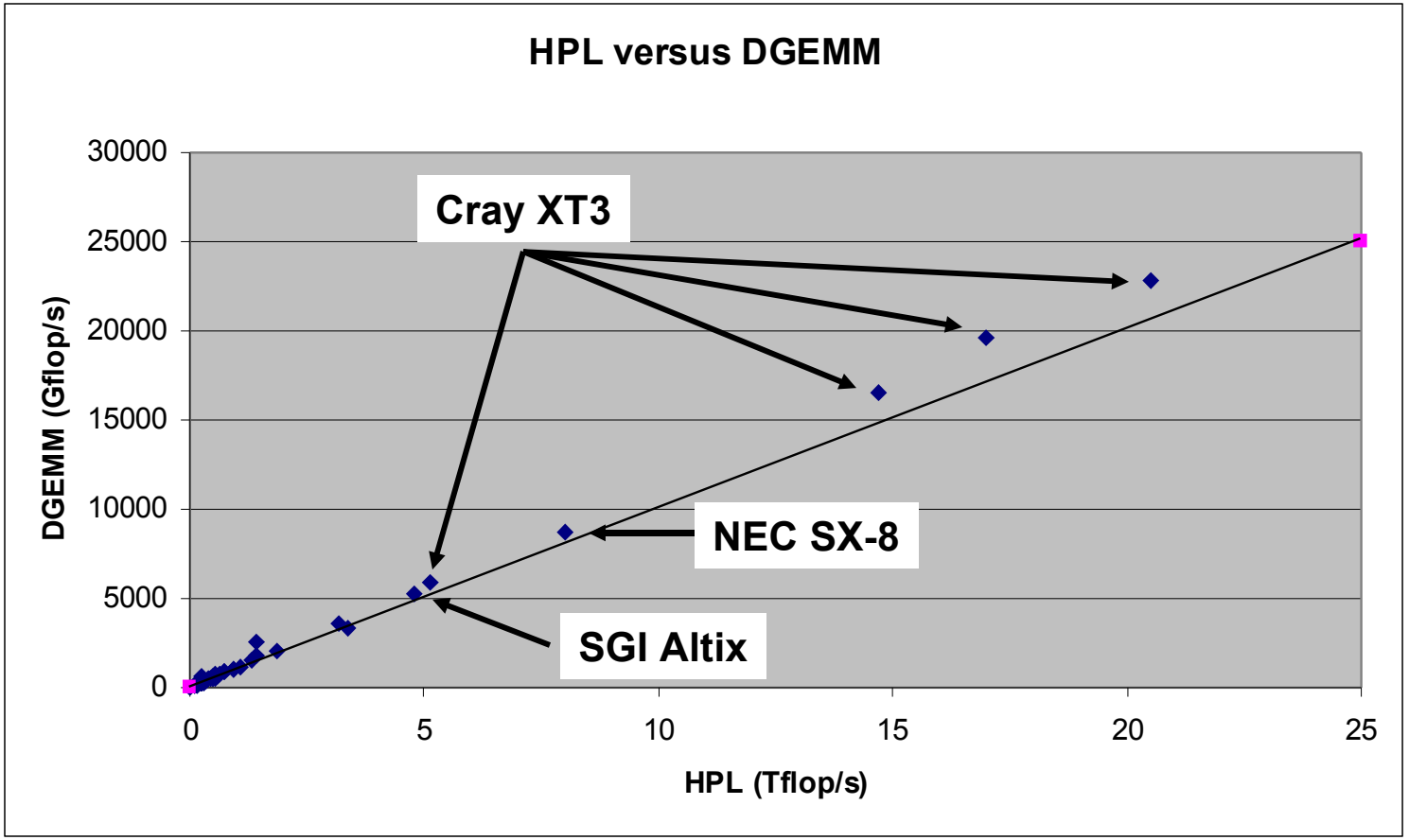
Good: can compare systems with different number of processors.
Bad: it's easy to scale peak and harder to scale real calculation.

Correlation: HPL vs. Theoretical Peak



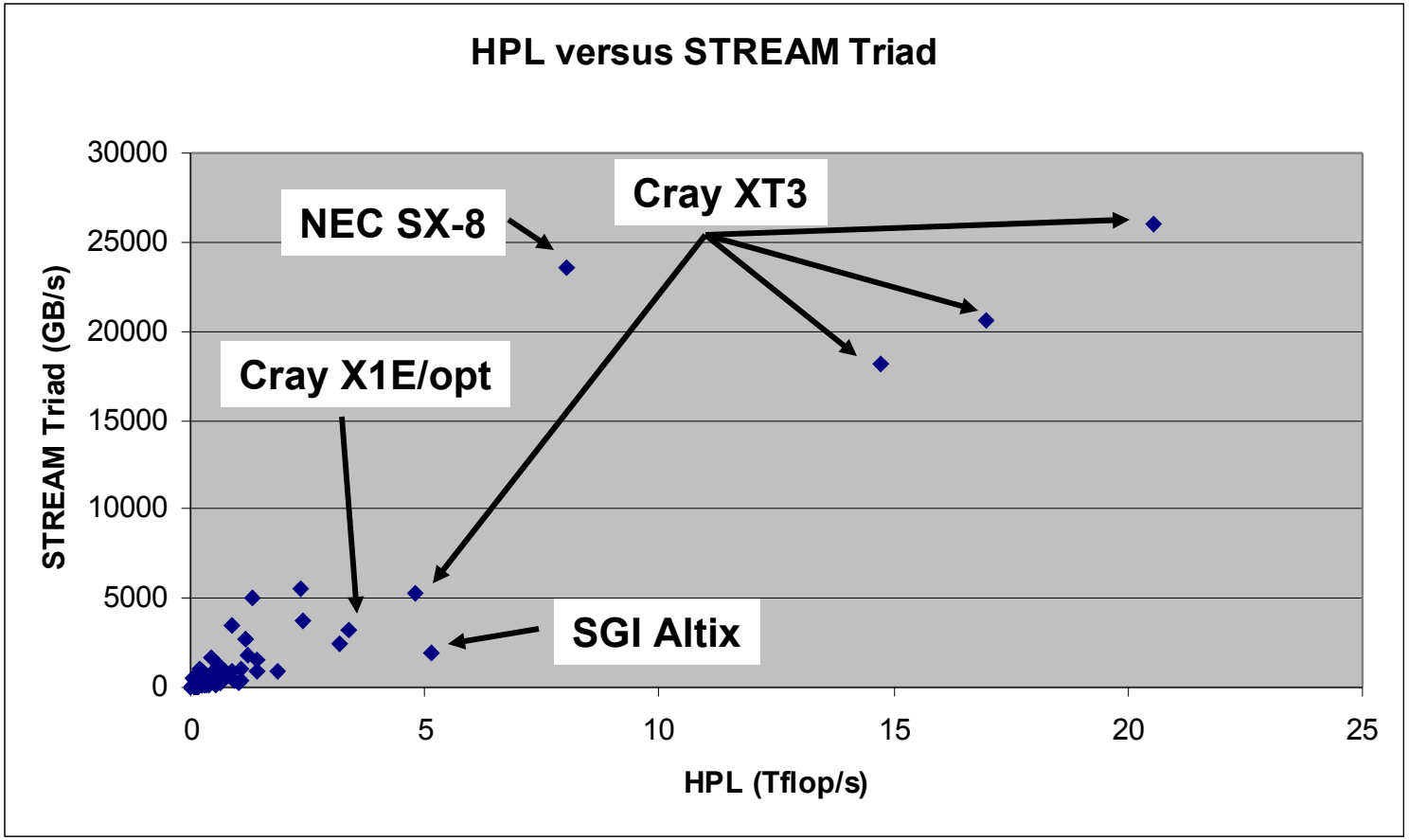
- How well does HPL data correlate with theoretical peak performance?

Correlation: HPL vs. DGEMM



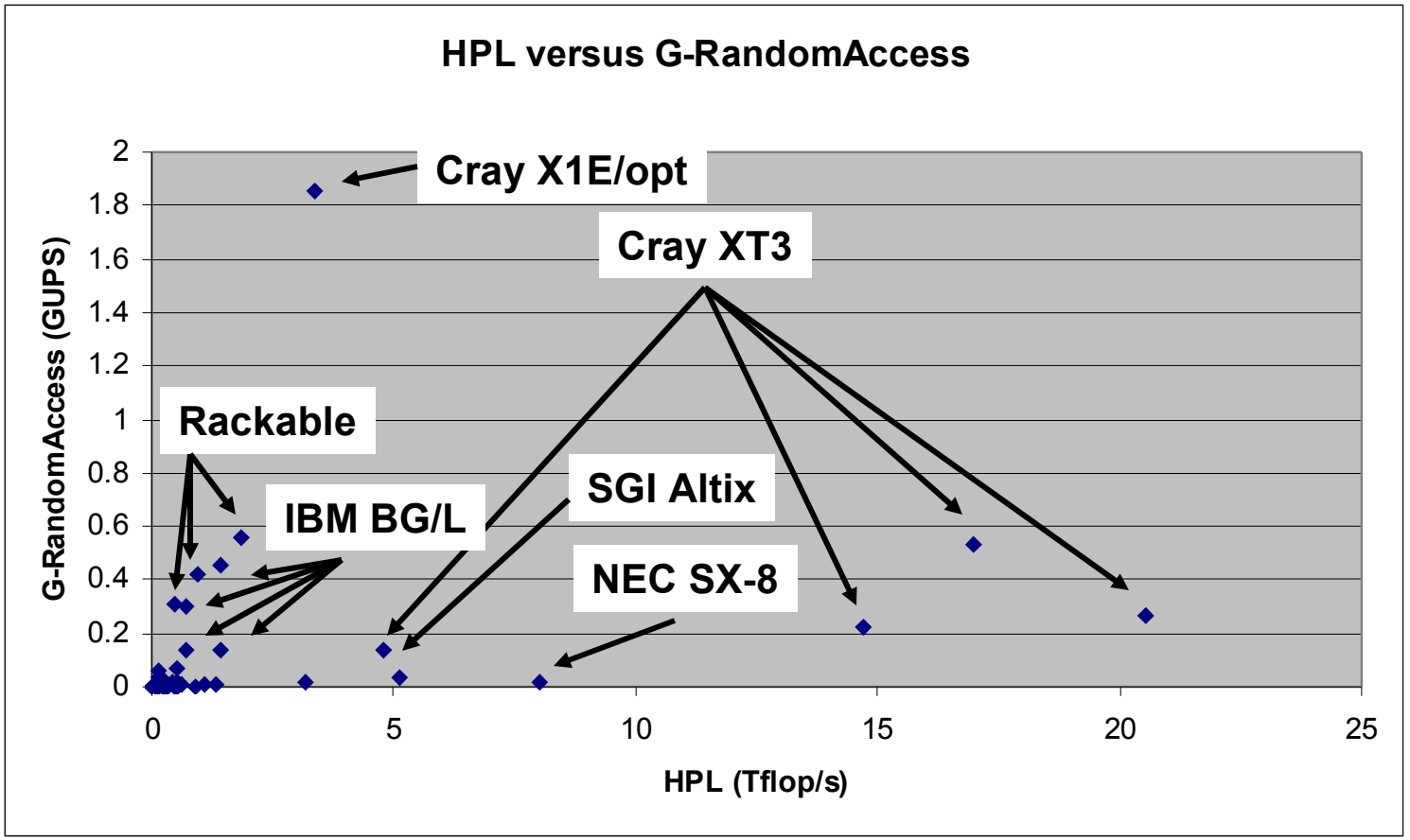
- Can I Run Just Run DGEMM Instead of HPL?
- DGEMM alone overestimates HPL performance
- Note the 1,000x difference in scales! (Tera/Giga)
- Exercise: correlate HPL versus Processors*DGEMM

Correlation: HPL vs. STREAM-Triad



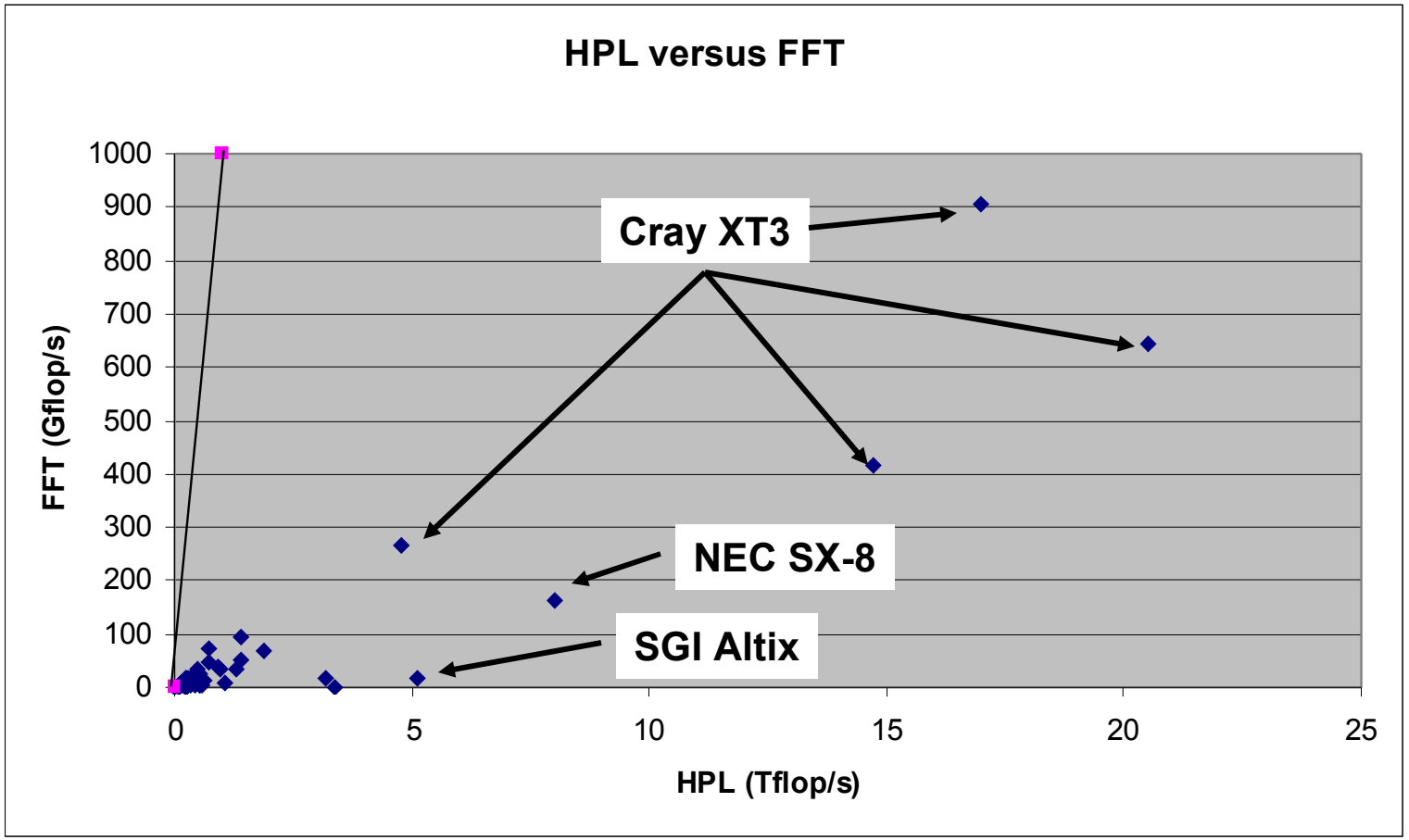
- How well does HPL correlate with STREAM-Triad performance?

Correlation: HPL vs. RandomAccess



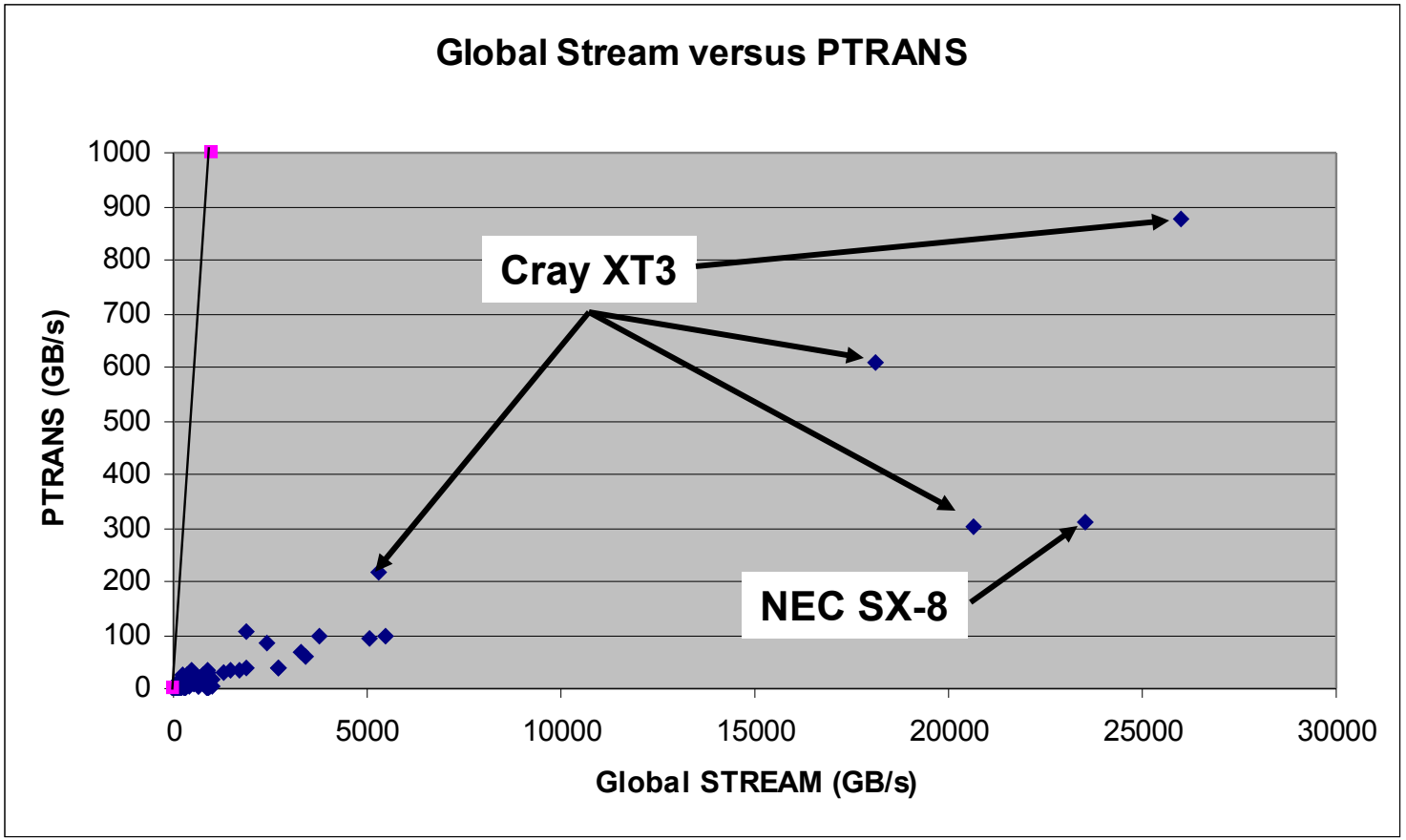
- How well does HPL correlate with G-RandomAccess performance?
- Note the 1,000x difference in scales! (Tera/Giga)

Correlation: HPL vs. FFT



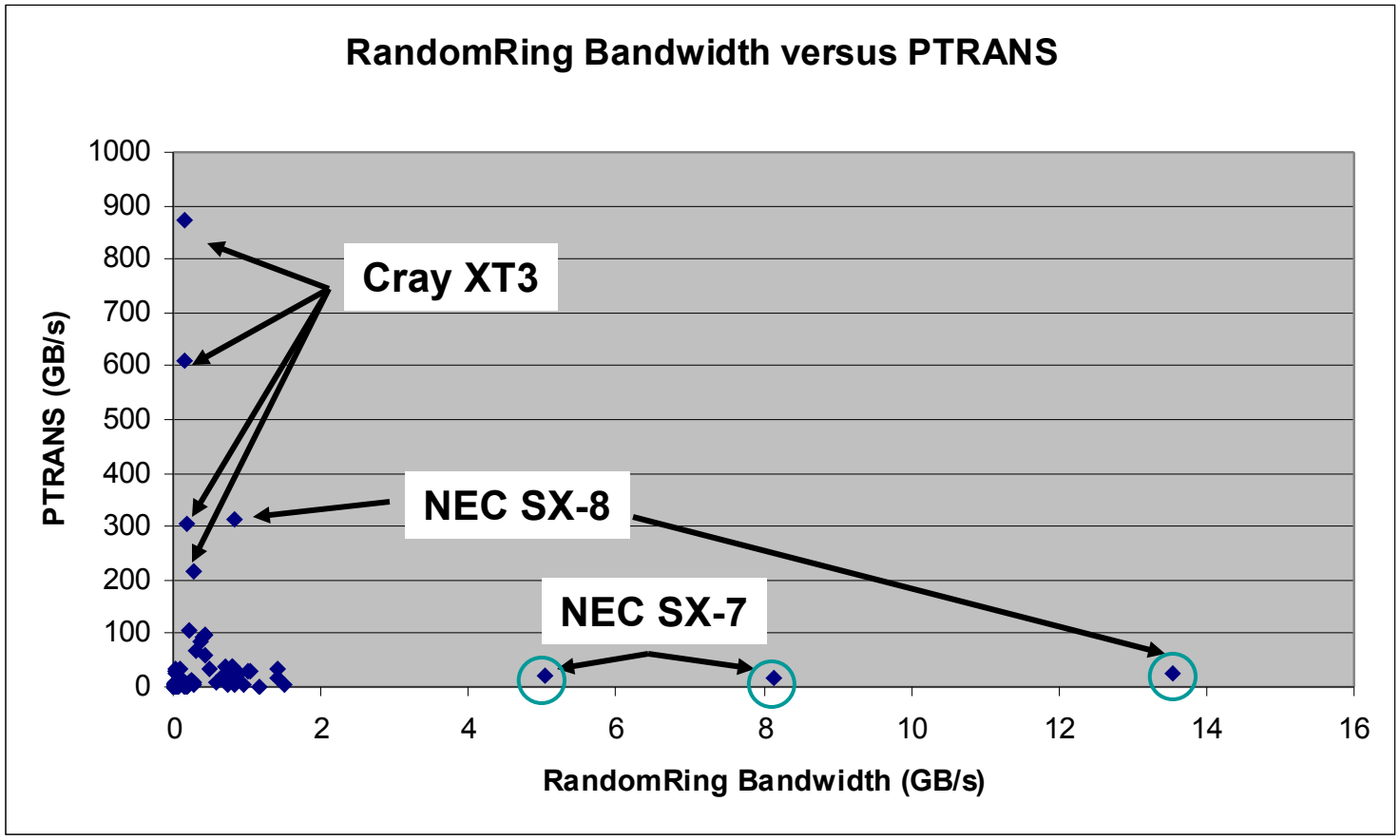
- How well does HPL correlate with FFT performance?
- Note the 1,000x difference in scales! (Tera/Giga)

Correlation: STREAM vs. PTRANS



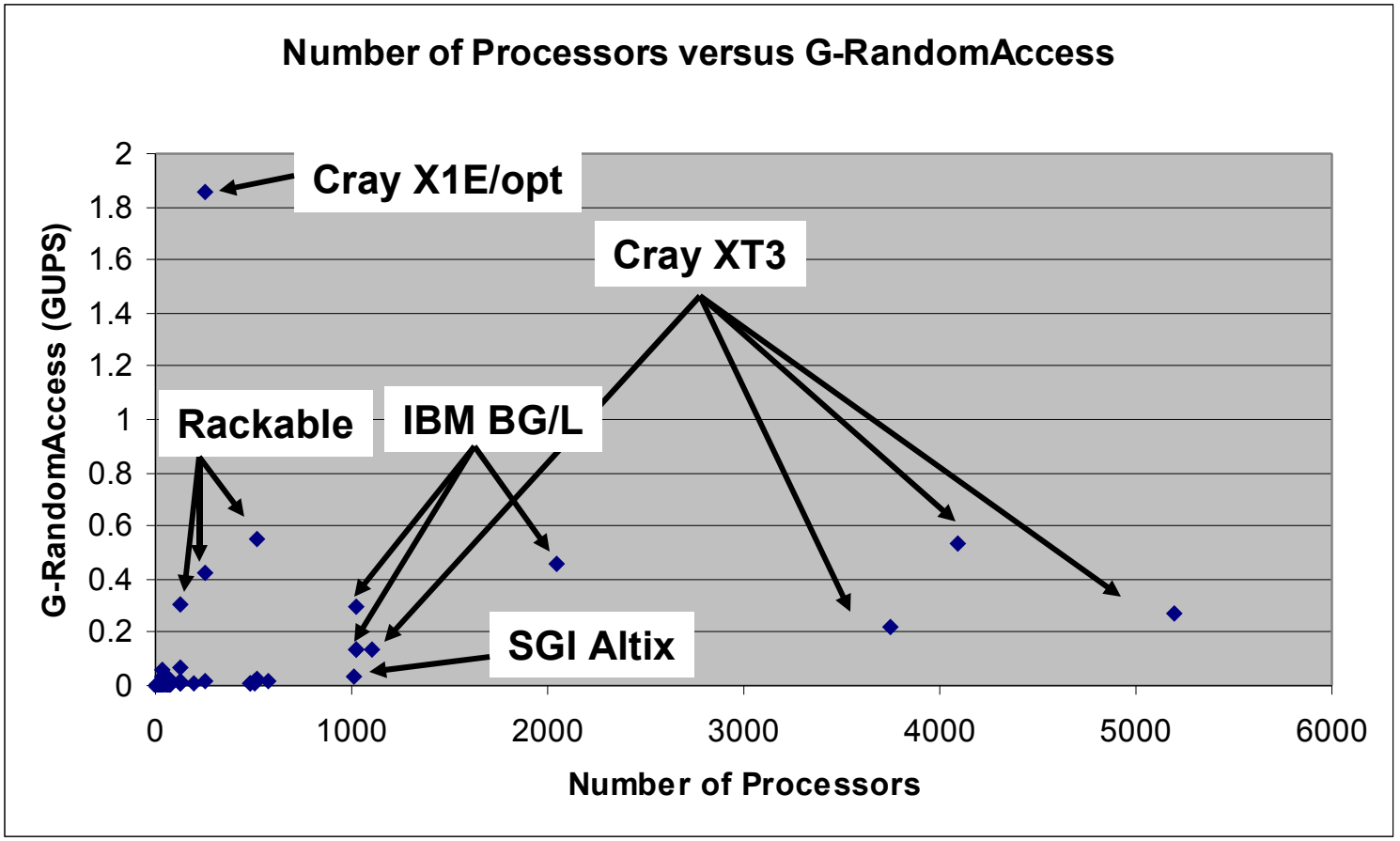
- How well does STREAM data correlate with PTRANS performance?

Correlation: RandomRing B/W vs. PTRANS



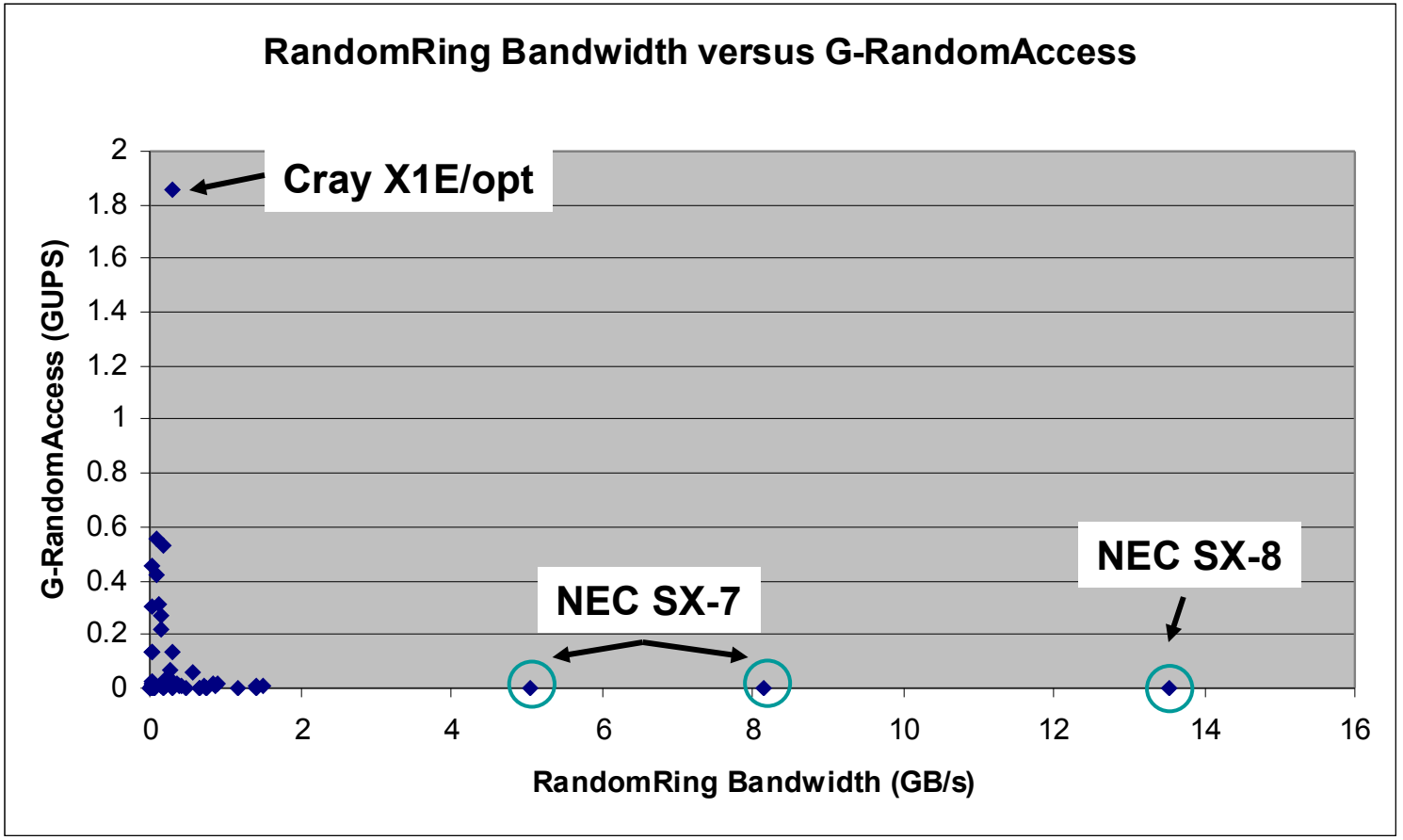
- How well does RandomRing Bandwidth data correlate with PTRANS performance
- Possible bad data?

Correlation: #Processors vs. RandomAccess



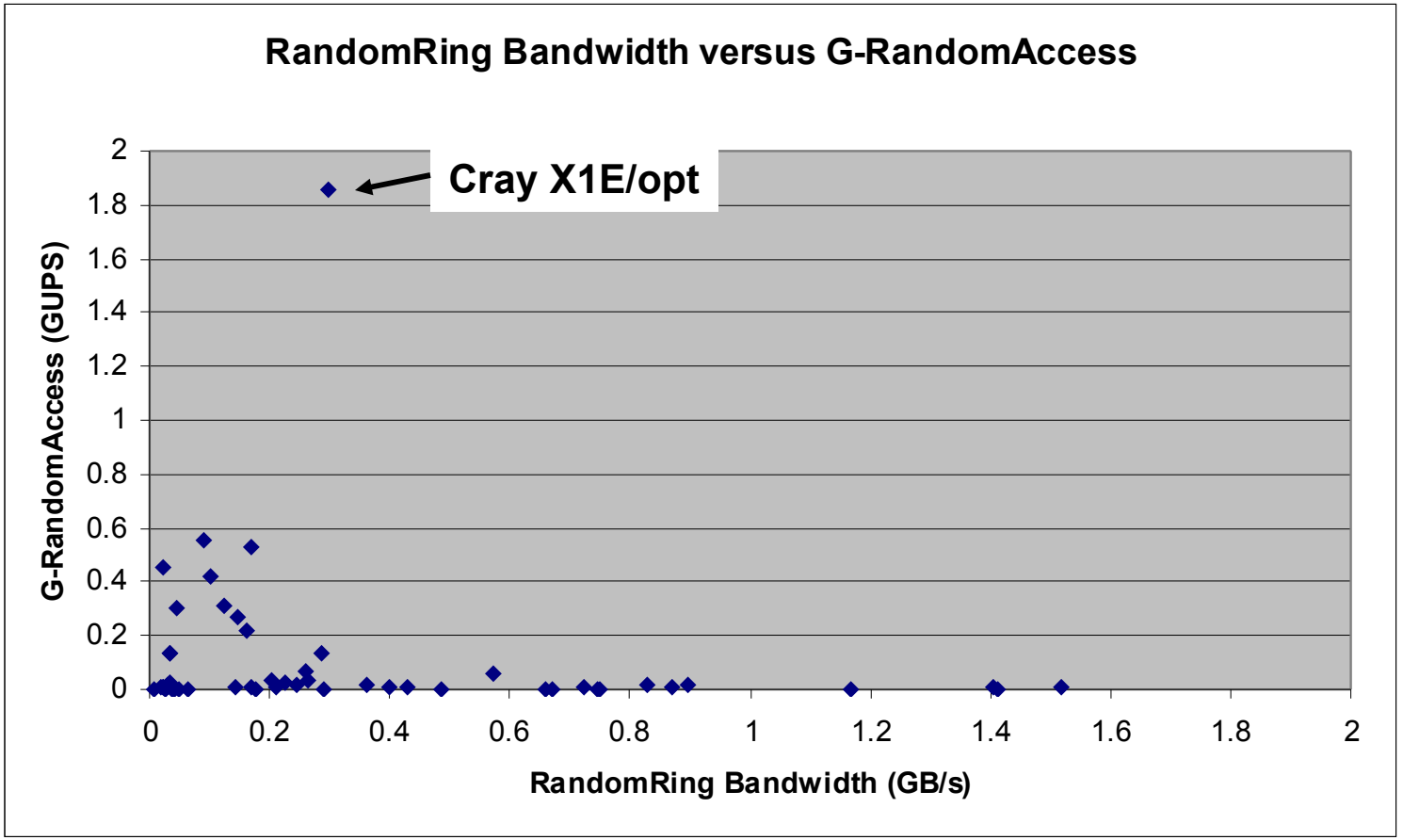
- Does G-RandomAccess scale with the number of processors?

Correlation: Random-Ring vs. RandomAccess



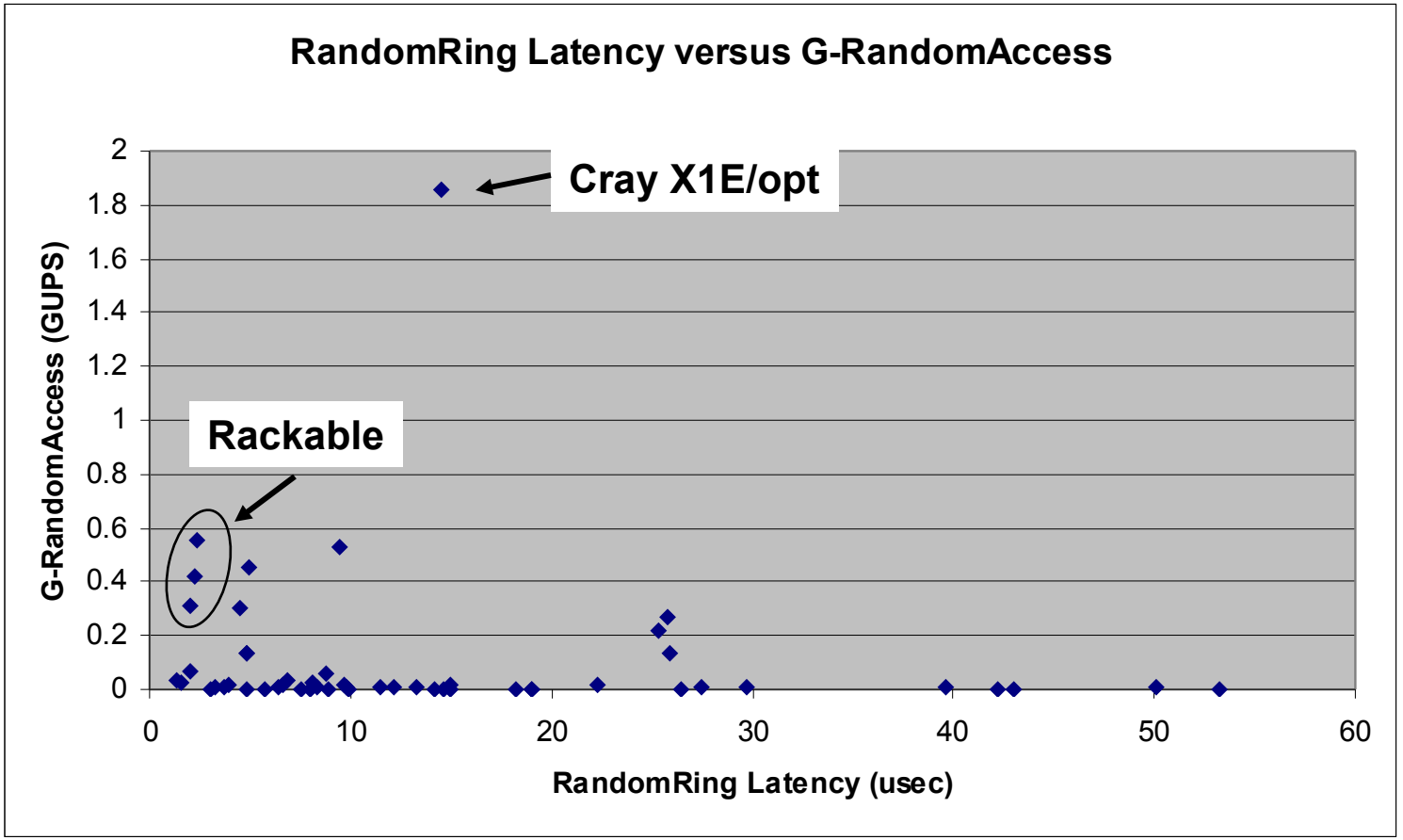
- Does G-RandomAccess scale with the RandomRing Bandwidth?
- Possible bad data?

Correlation: Random-Ring vs. RandomAccess



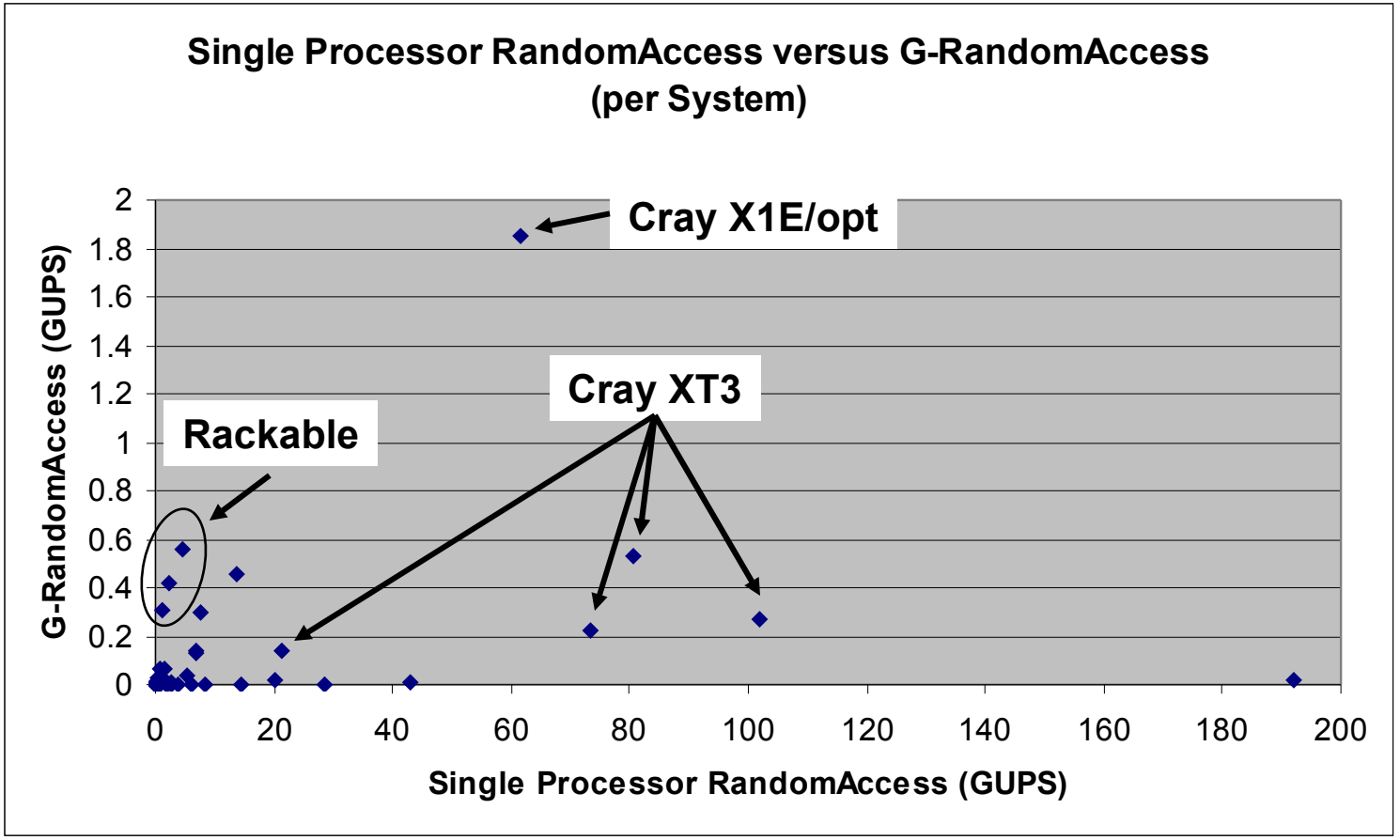
- Does G-RandomAccess scale with RandomRing Bandwidth?
- Ignoring possible bad data…

Correlation: Random-Ring vs. RandomAccess



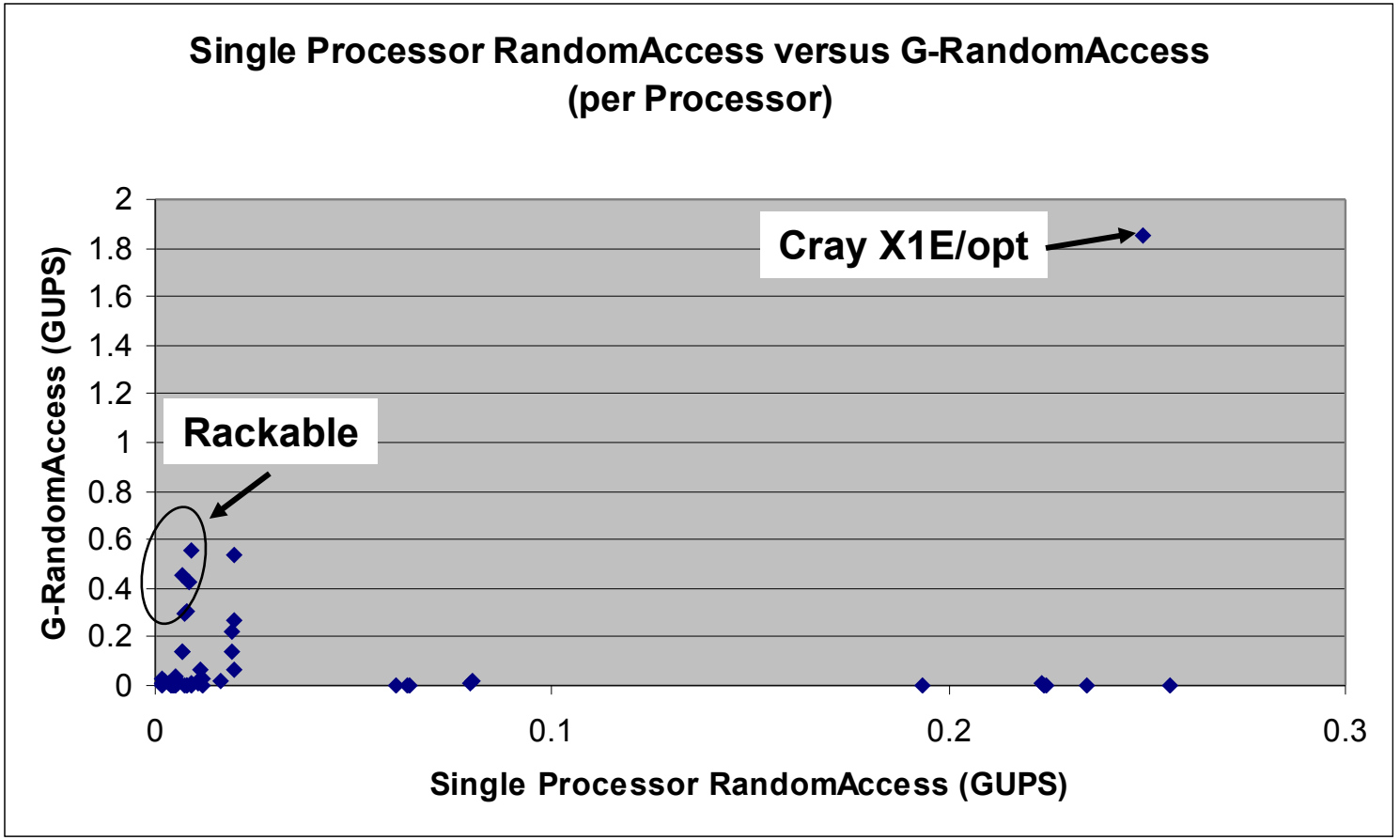
- Does G-RandomAccess scale with RandomRing Latency ?

Correlation: RandomAccess Local vs. Global



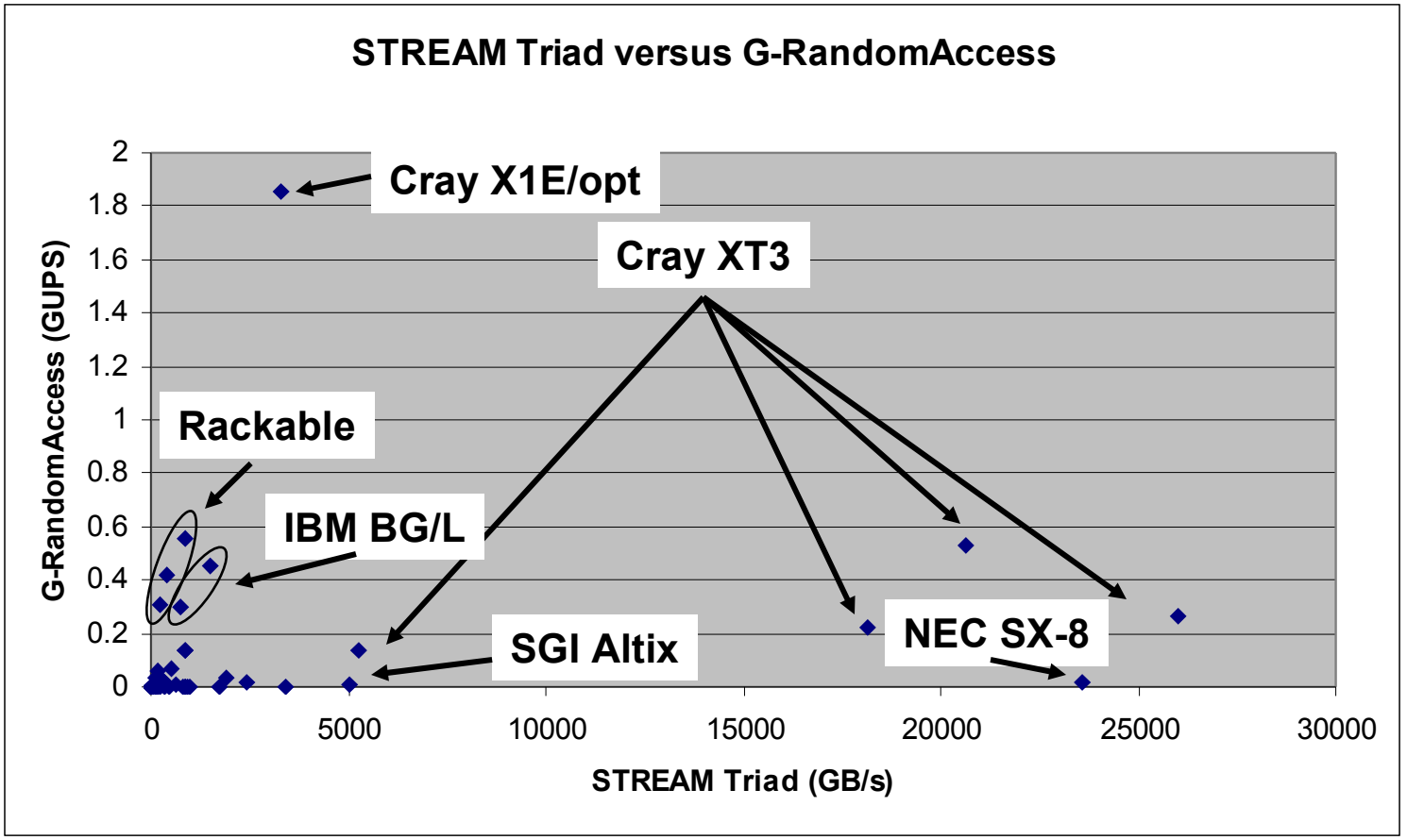
- Does G-RandomAccess scale with single processor RandomAccess performance (per system)?

Correlation: RandomAccess Local vs. Global



- Does G-RandomAccess scale with single processor RandomAccess performance?

Correlation:STREAM-Triad vs.RandomAccess



- Does G-RandomAccess scale with STREAM Triad?

RandomAccess Correlations Summary

- G-RandomAccess doesn't correlate with other tests
- Biggest improvement comes from code optimization
  - Limit on parallel look-ahead forces short messages
    - Typical MPI performance killer
    - Communication/computation overlap is wasted by message handling overhead
  - UPC implementation can be integrated with existing MPI code base to yield orders of magnitude speedup
  - Using interconnect topology and lower-level (less overhead) messaging layer is also a big win
    - Generalization from 3D torus: hyper-cube algorithm

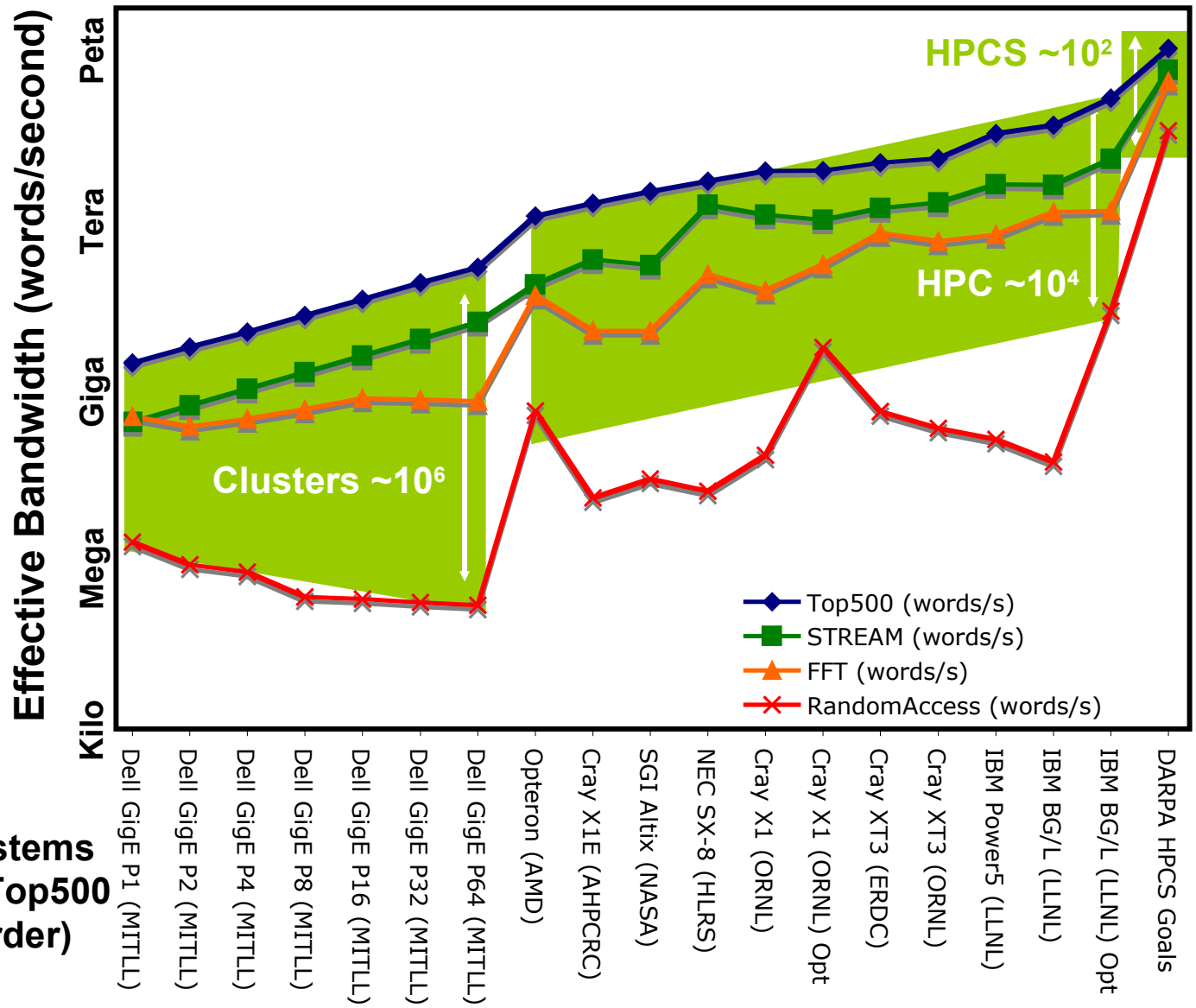
Principal Component Analysis

- Correlates all the tests simultaneously
- Load vectors hint at relationship between original results and the principal components
- Initial analysis:
  - 29 tests
    - Rpeak, HPL, PTRANS (x1)
    - DGEMM (x2)
    - FFT, RandomAccess (x3)
    - STREAM (x8)
    - b_eff (x10)
  - 103 computers
    - Must have all 29 valid (up-to-date) values
- Principal components (eigenvalues of covariance matrix of zero-mean, unscaled data):
  - 4.57, 1.17, 0.41, 0.15, 0.0085, 0.0027

- Only 3 (4) components
  - redundancy or
  - opportunity to check?

Effective Bandwidth Analysis

- All results in words/second
- Highlights memory hierarchy
- Clusters
  - Hierarchy steepens
- HPC systems
  - Hierarchy constant
- HPCS Goals
  - Hierarchy flattens
  - Easier to program



Public Web Resources

- Main HPCC website
  - http://icl.cs.utk.edu/hpcc/
- HPCC Awards
  - http://www.hpcchallenge.org/
- HPL
  - http://www.netlib.org/benchmark/hpl/
- STREAM
  - http://www.cs.virginia.edu/stream/
- PTRANS
  - http://www.netlib.org/parkbench/html/matrix-kernels.htr
- FFTE
  - http://www.fftw.jp/

HPCC Makefile Stucture (1/2)

- Sample Makefiles live in `hpl/setup`
- BLAS
  - `LAdir` – BLAS top directory for other `LA`-variables
  - `LAinc` – where BLAS headers live (if needed)
  - `LAlib` – where BLAS libraries live (`libmpi.a` and friends)
  - `F2CDEFS` – resolves Fortran-C calling issues (BLAS is usually callable from Fortran)
    - `-DAdd_, -DNoChange, -DUpCase, -Dadd__`
    - `-DStringSunStyle, -DStringStructPtr, -DStringStructVal, -DStringCrayStyle`
- MPI
  - `MPdir` – MPI top directory for other `MP`-variables
  - `MPinc` – where MPI headers live (`mpi.h` and friends)
  - `MPlib` – where MPI libraries live (`libmpi.a` and friends)

HPCC Makefile Stucture (1/2)

- Compiler
  - `CC` – C compiler
  - `CCNOOPT` – C flags without optimization (for optimization-sensitive code)
  - `CCFLAGS` – C flags with optimization
- Linker
  - `LINKER` – program that can link BLAS and MPI together
  - `LINKFLAGS` – flags required to link BLAS and MPI together
- Programs/commands
  - `SHELL, CD, CP, LN_S, MKDIR, RM, TOUCH`
  - `ARCHIVER, ARFLAGS, RANLIB`

MPI Implementations for HPCC

- Vendor
  - Cray (MPT)
  - IBM (POE)
  - SGI (MPT)
  - Dolphin, Infiniband (Mellanox, Voltaire, ...), Myricom (GM, MX), Quadrics, PathScale, Scali, ...
- Open Source
  - MPICH1, MPICH2 (http://www-unix.mcs.anl.gov/mpi/mpich/)
  - Lam MPI (http://www.lam-mpi.org/)
  - OpenMPI (http://www.open-mpi.org/)
  - LA-MPI (http://public.lanl.gov/lampi/)
- MPI implementation components
  - Compiler (adds MPI header directories)
  - Linker (need to link in Fortran I/O)
  - Exe (poe, mprun, mpirun, aprun, mpiexec, ...)

Fast BLAS for HPCC

- Vendor
  - AMD (AMD Core Math Library)
  - Cray (SciLib)
  - HP (MLIB)
  - IBM (ESSL)
  - Intel (Math Kernel Library)
  - SGI (SGI/Cray Scientific Library)
  - ...
- Free implementations
  - ATLAS
    http://www.netlib.org/atlas/
  - Goto BLAS
    http://www.cs.utexas.edu/users/f

    http://www.tacc.utexas.edu/resources/software
- Implementations that use Threads
  - Some vendor BLAS
  - Atlas
  - Goto BLAS
- You should never use reference BLAS from Netlib
  - There are better alternatives for every system in existence

Tuning Process: Internal to HPCC Code

- Changes to source code are not allowed for submission
- But just for tuning it's best to change a few things
  - Switch off some tests temporarily
- Choosing right parallelism levels
  - Processes (MPI)
  - Threads (OpenMP in code, vendor in BLAS)
  - Processors
  - Cores
- Compile time parameters
  - More details below
- Runtime input file
  - More details below

Tuning Process: External to HPCC Code

- MPI settings examples
  - Messaging modes
    - Eager polling is probably not a good idea
  - Buffer sizes
  - Consult MPI implementation documentation
- OS settings
  - Page size
    - Large page size should be better on many systems
  - Pinning down the pages
    - Optimize affinity on DSM architectures
  - Priorities
  - Consult OS documentation

Parallelism Examples with Comments

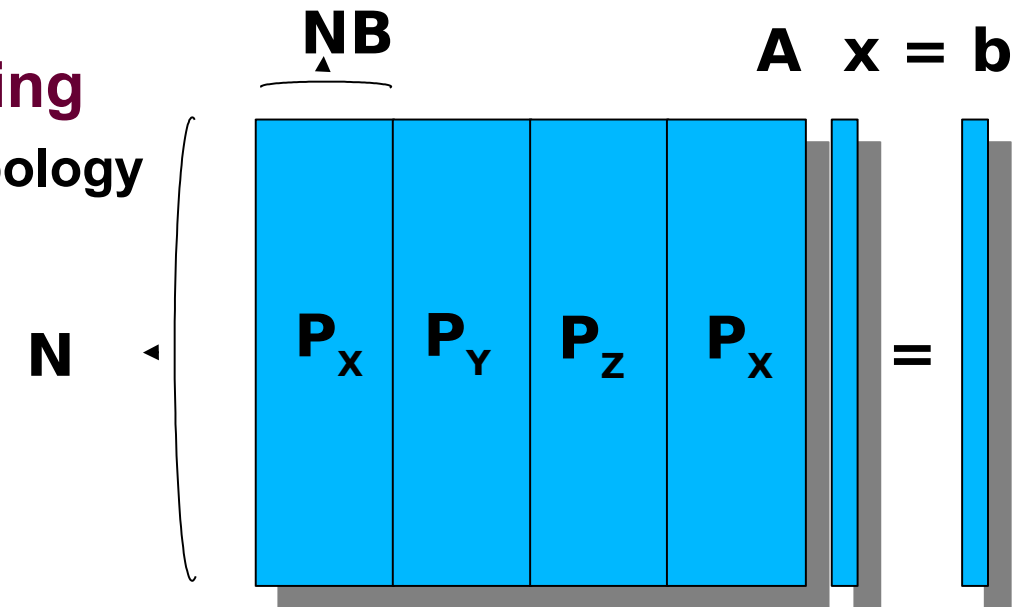
- Pseudo-threading helps but be careful
  - Hyper-threading
  - Simultaneous Multi-Threading
  - ...
- Cores
  - Intel (x86-64, Itanium), AMD (x86)
  - Cray: SSP, MSP
  - IBM Power4, Power5, ...
  - Sun SPARC
- SMP
  - BlueGene/L (single/double CPU usage per card)
  - SGI (NUMA, ccNUMA, DSM)
  - Cray, NEC
- Others
  - Cray MTA (no MPI !)

HPCC Input and Output Files

- Parameter file `hpccinf.txt`
  - HPL parameters
    - Lines 5-31
  - PTRANS parameters
    - Lines 32-36
  - Indirectly: sizes of arrays for all HPCC components
    - Hard coded
- Output file hpccoutf.txt
  - Must be uploaded to the website
  - Easy to parse
  - More details later...
- Memory file `hpccmemf.txt`
  - Memory available per MPI process `Process=64`
  - Memory available per thread `Thread=64`
  - Total available memory `Total=64`
  - Many HPL and PTRANS parameters might not be optimal

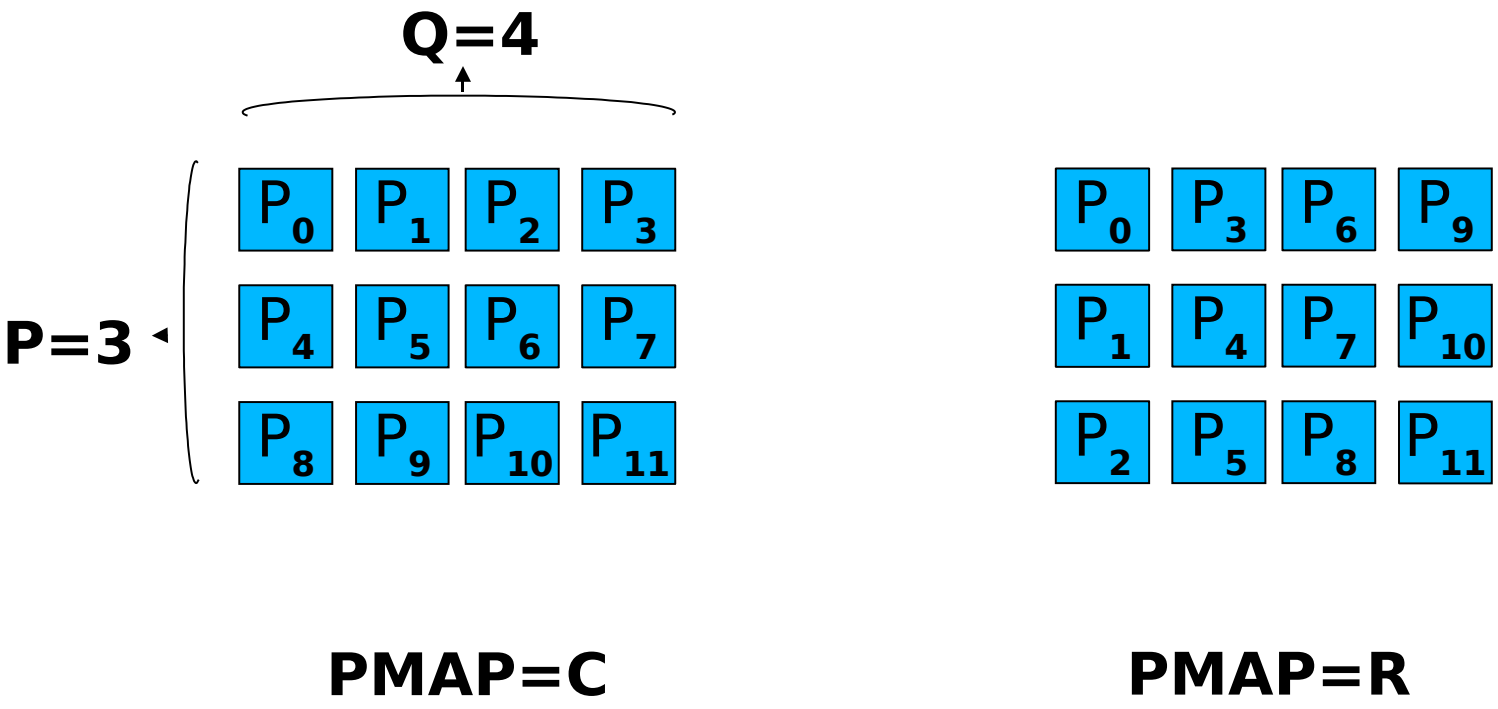
Tuning HPL: Introduction

- Performance of HPL comes from
  - BLAS
  - Input file hpccinf.txt
- Essential parameters in the input file
  - N – matrix size
  - NB – blocking factor
    - Influences BLAS performance and load balance
  - PMAP – process mapping
    - Depends on network topology
  - PxQ – process grid
- Definitions

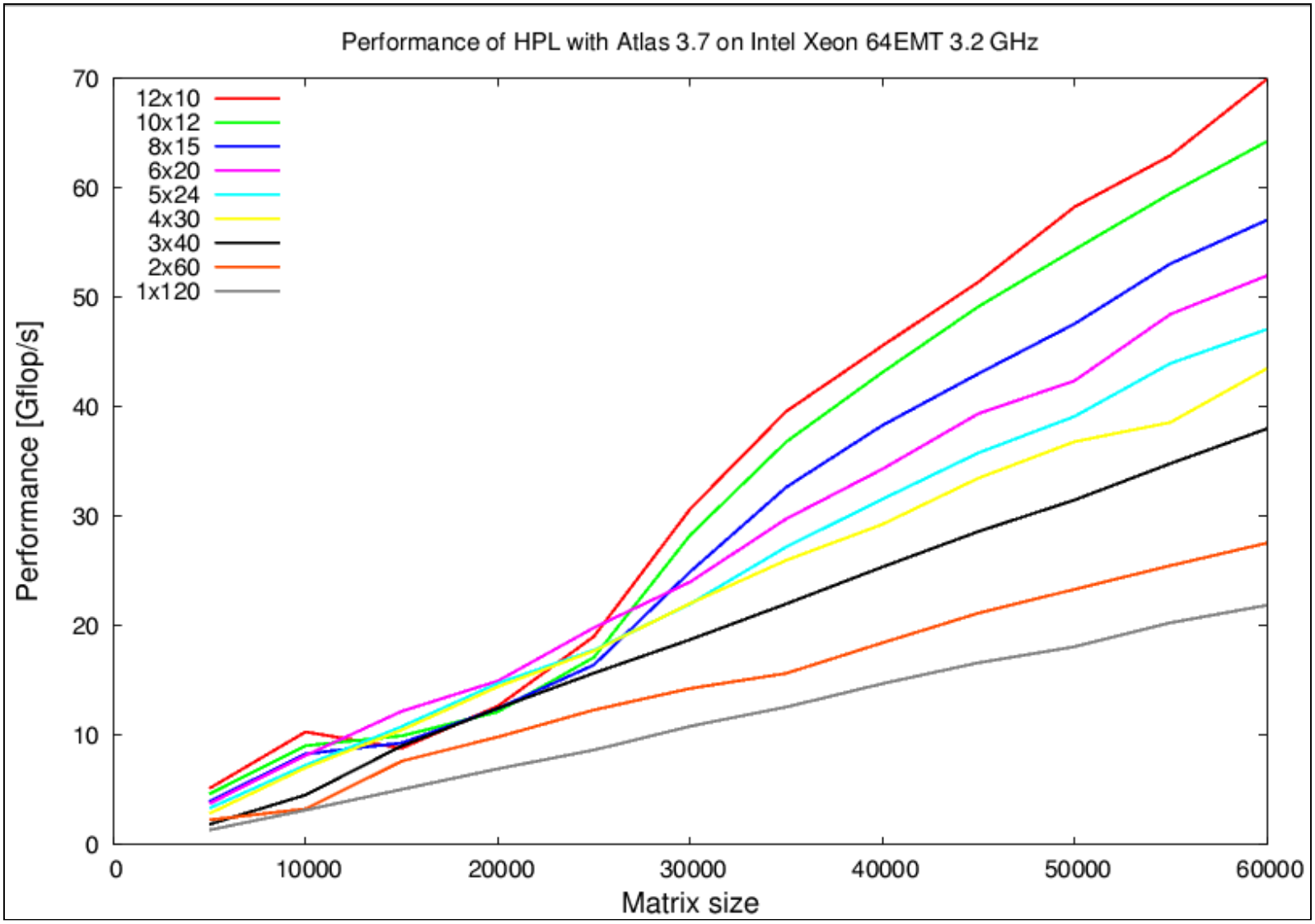


Tuning HPL: More Definitions

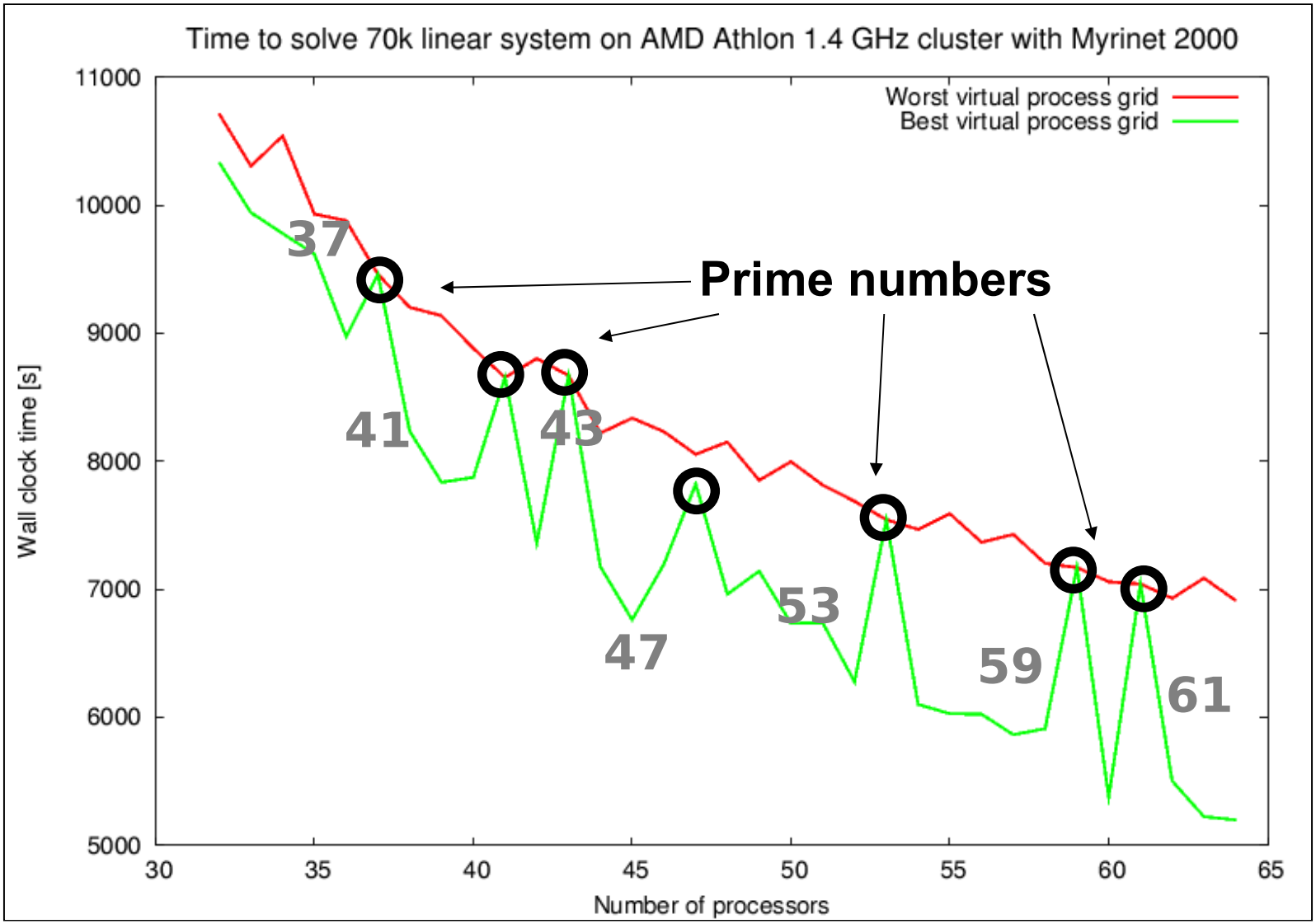
- **Process grid parameters: P, Q, and PMAP**



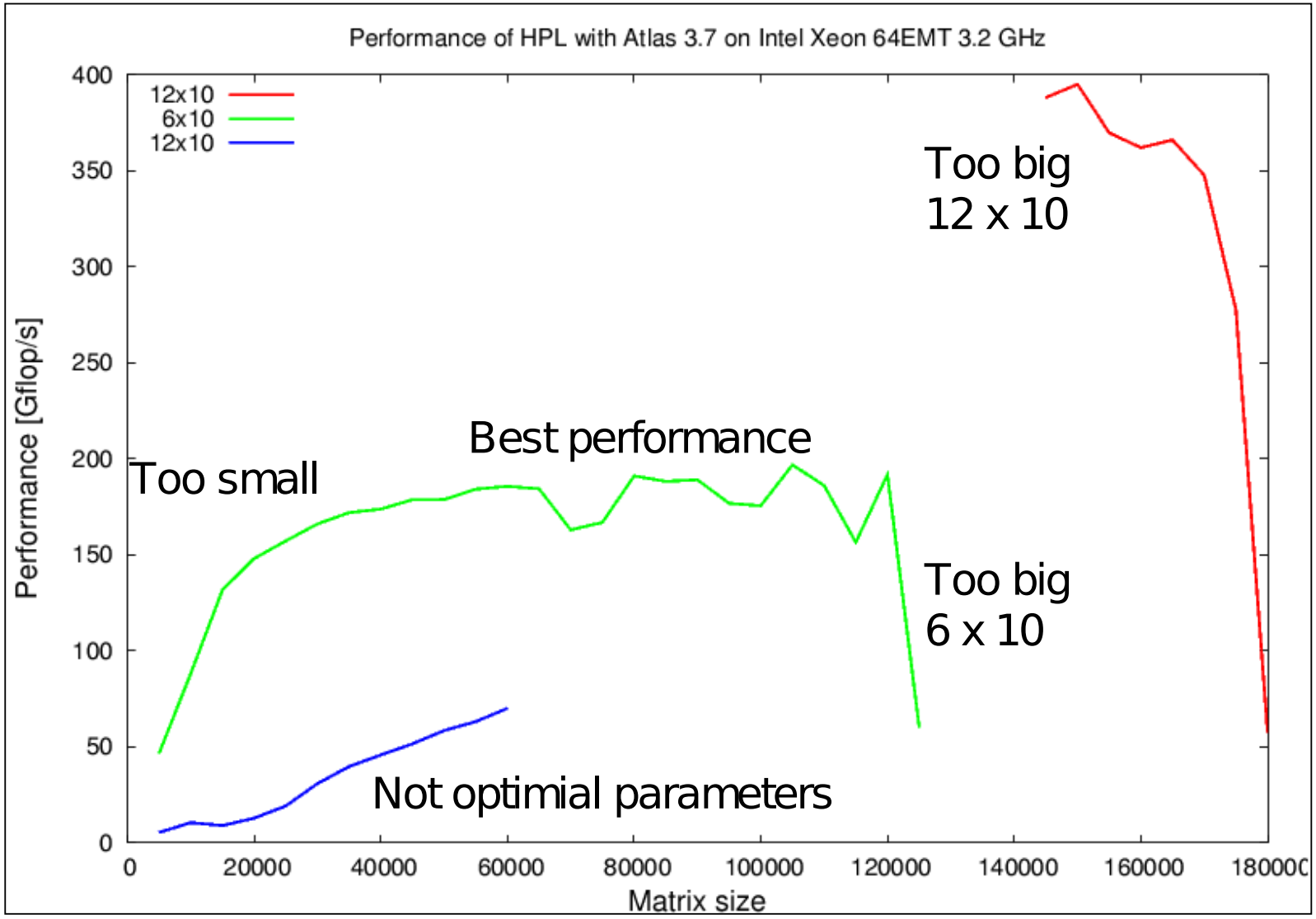
Tuning HPL: Selecting Process Grid



Tuning HPL: Selecting Number of Processors



Tuning HPL: Selecting Matrix Size



Tuning HPL: Further Details

- **Much more details from HPL's author:**
- **Antoine Petitet**
- **http://www.netlib.org/benchmark/hpl/**

Tuning FFT

- Compile-time parameters
  - `FFTE_NBLK`
    - blocking factor
  - `FFTE_NP`
    - padding (to alleviate negative cache-line effects)
  - `FFTE_L2SIZE`
    - size of level 2 cache
- Use FFTW instead of FFTE
  - Define `USING_FFTW` symbol during compilation
  - Add FFTW location and library to linker flags

Tuning STREAM

- Intended to measure main memory bandwidth
- Requires many optimizations to run at full hardware speed
  - Software pipelining
  - Prefetching
  - Loop unrolling
  - Data alignment
  - Removal of array aliasing
- Original STREAM has advantages
  - Constant array sizes (known at compile time)
  - Static storage of arrays (at full compiler's control)

Tuning PTRANS

- Parameter file `hpccinf.txt`
  - Line 33 — number of matrix sizes
  - Line 34 — matrix sizes
    - Must not be too small – enforced in the code
  - Line 35 — number of blocking factors
  - Line 36 — blocking factors
    - No need to worry about BLAS
    - Very influential for performance

Tuning b_eff

- b_eff (Effective bandwidth and latency) test can also be tuned
- Tuning must use only standard MPI calls
- Examples
  - Persistent communication
  - One-sided communication

HPCC Output File

- The output file has two parts
  - Verbose output (free format)
  - Summary section
    - Pairs of the form:
      `name=value`
- The summary section names
  - MPI* — global results
    - Example: MPIRandomAccess_GUPs
  - Star* — embarrassingly parallel results
    - Example: StarRandomAccess_GUPs
  - Single* — single process results
    - Example: SingleRandomAccess_GUPs

Optimized Submission Ideas

- For optimized run the same MPI harness has to be run on the same system
- Certain routines can be replaced – the timed regions
- The verification has to pass – limits data layout and accuracy of optimization
- Variations of the reference implementation are allowed (within reason)
  - No Strassen algorithm for HPL due to different operation count
- Various non-portable C directives can significantly boost performance
  - Example: `#pragma ivdep`
- Various messaging substrates can be used
  - Removes MPI overhead
- Various languages can be used
  - Allows for direct access to non-portable hardware features
  - UPC was used to increase RandomAccess performance by orders of magnitude
- Optimizations need to be explained upon results submission