
Simple and Composite Metrics for System Throughput in HPC

John D. McCalpin, Ph.D.
IBM Corporation
Austin, TX

Workshop on Performance Characterization,
Modelling and Benchmarking for HPC Systems
May 6, 2003



What Are We Doing Here?

- There are many different problems being addressed in this workshop – maybe too many

What Are We Doing Here?

- There are many different problems being addressed in this workshop – maybe too many
- **Some problems are relatively easy**
 - **Simple kernel benchmarks**

What Are We Doing Here?

- There are many different problems being addressed in this workshop – maybe too many
- Some problems are relatively easy
- **Some problems are hard, but tractable**
 - **Application-specific performance models**

What Are We Doing Here?

- There are many different problems being addressed in this workshop – maybe too many
- Some problems are relatively easy
- Some problems are hard, but tractable
- **Some problems are intractable, but too important to give up on**
 - What is the relative “science value” of System A vs System B?



Topics of Interest

- What is the correct arithmetical calculus for performance evaluation?

Topics of Interest

- What is the correct arithmetical calculus for performance evaluation?
- How general (or how specific) do we want this analysis to be?

Topics of Interest

- What is the correct arithmetical calculus for performance evaluation?
- How general (or how specific) do we want this analysis to be?
- How much is it going to bother us when we are wrong?

The Calculus of Performance Modelling

- This calculus looks a lot like an extension of Amdahl's Law
 - Amdahl's Law is really a very narrow special case of the more general calculus
- Key Concepts
 - Lots of “stuff” happens in parallel on independent pieces of the system
 - The amount of overlap across this parallel “stuff” is variable and can be nearly impossible to predict

What parallelism am I talking about?

- This is not necessarily “parallelism” in the Amdahl’s Law sense
- Amdahl’s Law is a special case
 - Assumes homogeneity of parallel work
 - Assumes no overlap between serial and parallel work
 - Assumes no additional parallel overheads

What parallelism am I talking about?

- Examples of parallel hardware components
 - Floating-point functional units
 - Load/store units
 - Integer functional units
 - Branch units
 - Register file(s)
 - L1, L2, L3 (...) cache miss handling control structures (I & D)
 - Wires/busses connecting levels of the memory hierarchy
 - L1, L2, L3, memory RAM banks
 - Virtual address translation facilities
- This is already more than 10 degrees of parallelism for each processor for most current architectures
 - A “balanced” code has lots of potential for overlap – or not



Performance ? 1/Time

- Time = Work/Rate
- Repeat for each component: $T_i = W_i/R_i$

Performance ? 1/Time

- Time = Work/Rate
- Repeat for each component: $T_i = W_i/R_i$
- Big Issues:
 - Where do we get the W_i 's?
 - Can we understand the R_i 's well enough to be useful?
 - How do we combine the T_i 's?

Performance ? 1/Time

- Time = Work/Rate
- Repeat for each component: $T_i = W_i/R_i$
- Big Issues:
 - Where do we get the W_i 's?
 - Can we understand the R_i 's well enough to be useful?
 - How do we combine the T_i 's?
- This talk will mostly address this last issue

Arithmetic Formulation of the Calculus

- The Calculus is based on a new arithmetic operator, called “plus or max” – I will use the infix symbol “\$” here

- Definition

$$\text{Max}(T_1, T_2) = T_1 \$ T_2 = T_1 + T_2$$

- Assume:

- $W_1/R_1 = T_1$ when executed in isolation
- $W_2/R_2 = T_2$ when executed in isolation

- Q: How long does it take to perform W_1 and W_2 at the same time?

- A: $T_{\text{total}} = T_1 \$ T_2$

- I assert that this holds true for most sets of “independent” operations on most computer systems



Consequences of the Calculus

- Note that “=” now means a range of values
- The “plus or max” operator is associative and commutative, just like normal addition
- The composite operator can be written:

$$\text{MAX}_{i=1}^N (T_i) = \text{\$} T_i = \text{\$}_{i=1}^N T_i$$

Bounds

$$\sum_{i=1}^N T_i \leq N \cdot \max_{i=1}^N T_i$$

- Note that the ratio of upper bound to lower bound is equal to the number of degrees of parallelism in the model
- In reality, the ratio is equal to the number of degrees of parallelism in the **system** !
 - This number can be uncomfortably large
 - Unbalanced systems make for easier performance projections



A Dilemma

-
- Parallelism is essential for increasing potential performance
 - Every generation of systems has more parallelism and more kinds of parallelism
 - It is being increased by the vendors whether you want it or not
 - Parallelism also increases the uncertainty of performance models
 - This is intrinsic to the performance calculus
 - It is exacerbated by the lack of transparency in the amount of overlap possible between parallel “stuff”
 - This is inevitable as systems get more and more complex



The Dilemma, continued

- Recall that the parallelism considered here is the parallelism of the real hardware, not the parallelism of the model
 - Reducing the complexity of the model does not help!
 - But it gets worse...
- The Competitive Benchmarking Hypothesis:
For any system with N degrees of parallelism, at least one application can be found that is limited in performance by each and every one of those N degrees
- Consequence:
For any “reduced” performance model, errors in performance projection of $O(1)$ are guaranteed for some set of applications



We still need to deal with Horst's question

- How do we quantify the relative “performance value” of various systems for “general” scientific research?
- Maybe someday we can accumulate enough data about enough applications to make credible estimates of the distributions of the W_i 's by application area, but I remain skeptical of the potential for non-application-specific modelling...
- For now we need to wing it^{H^H^H^H^H^H} apply heuristics



Current Status for Composite Performance Metrics

- We don't have an exhaustive list of the parallel functional units for each system (definitely not a portable list)

Current Status for Composite Performance Metrics

- We don't have an exhaustive list of the parallel functional units for each system (definitely not a portable list)
- We don't know the detailed formulae for the R_i 's for each of those functional units

Current Status for Composite Performance Metrics

- We don't have an exhaustive list of the parallel functional units for each system (definitely not a portable list)
- We don't know the detailed formulae for the R_i 's for each of those functional units
- We don't have the W_i 's to calculate the T_i 's



Current Status for Composite Performance Metrics

- We don't have an exhaustive list of the parallel functional units for each system (definitely not a portable list)
- We don't know the detailed formulae for the R_i 's for each of those functional units
- We don't have the W_i 's to calculate the T_i 's
- We would not be able to figure out how the T_i 's overlap (or not) anyway

Current Status for Composite Performance Metrics

- We don't have an exhaustive list of the parallel functional units for each system (definitely not a portable list)
- We don't know the detailed formulae for the R_i 's for each of those functional units
- We don't have the W_i 's to calculate the T_i 's
- We would not be able to figure out how the T_i 's overlap (or not) anyway
- Other than that, we are in great shape!



So what to do?

- We cannot handle all of the possible bottlenecks
- So we take the most common bottlenecks and rank order them
- Data to build these rankings is still mostly based on “educated guess”

Methodology from 15,240 meters

- Pick a workload of interest & gather performance characteristics
- Start with the most common bottleneck, estimate the W_i , build a model $T_i = W_i/R_i$
- Add the next most common bottleneck, estimate the W_i , and ponder the overlap issues
- Repeat until you cannot reduce the error of the projections any more

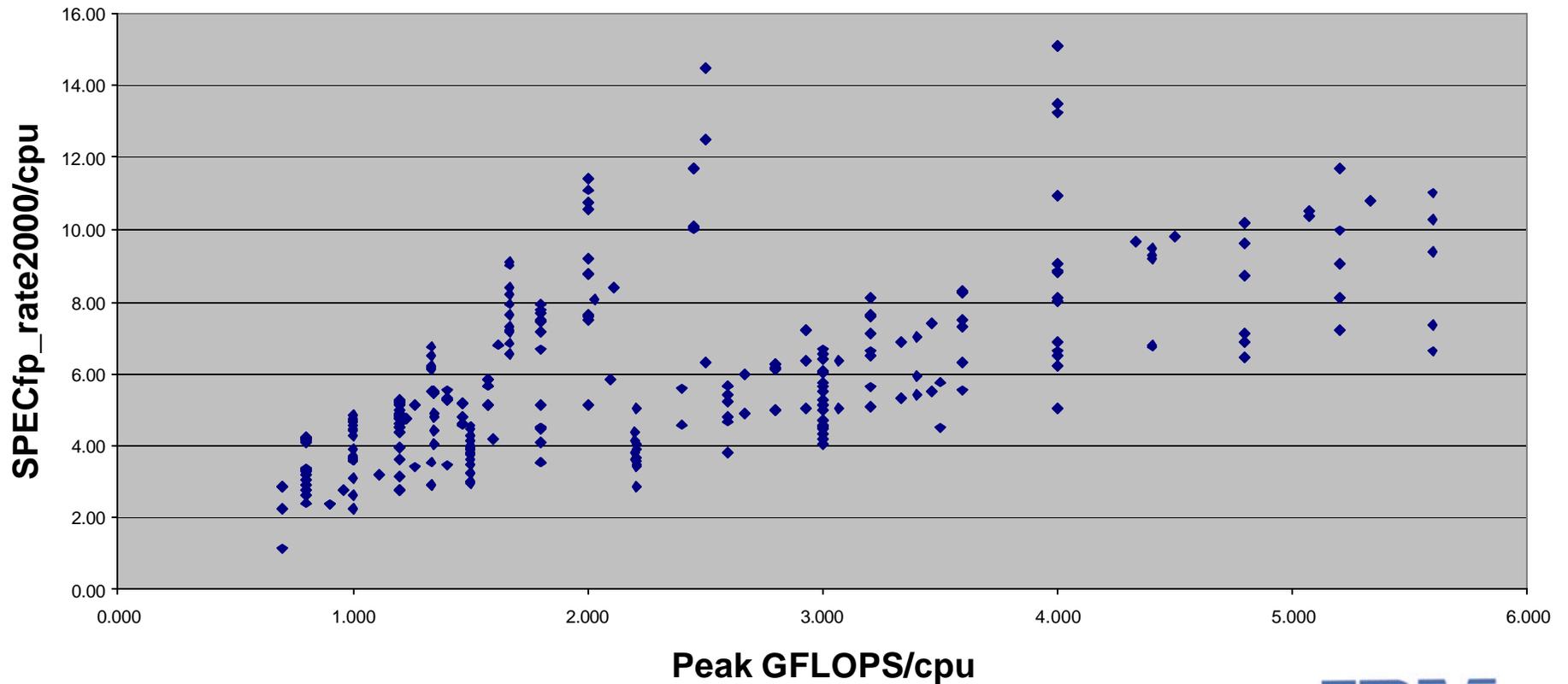
A Simple Constructive Methodology

- The target workload is SPECfp_rate2000
 - All 331 published values as of November 12, 2002
- Assume that FP arithmetic is the primary bottleneck
- Add memory bandwidth as the secondary bottleneck
- No W_i 's were measured
 - model values were obtained *a posteriori* by modifying the parameters of a simple analytic model to minimize the RMS error of the projections



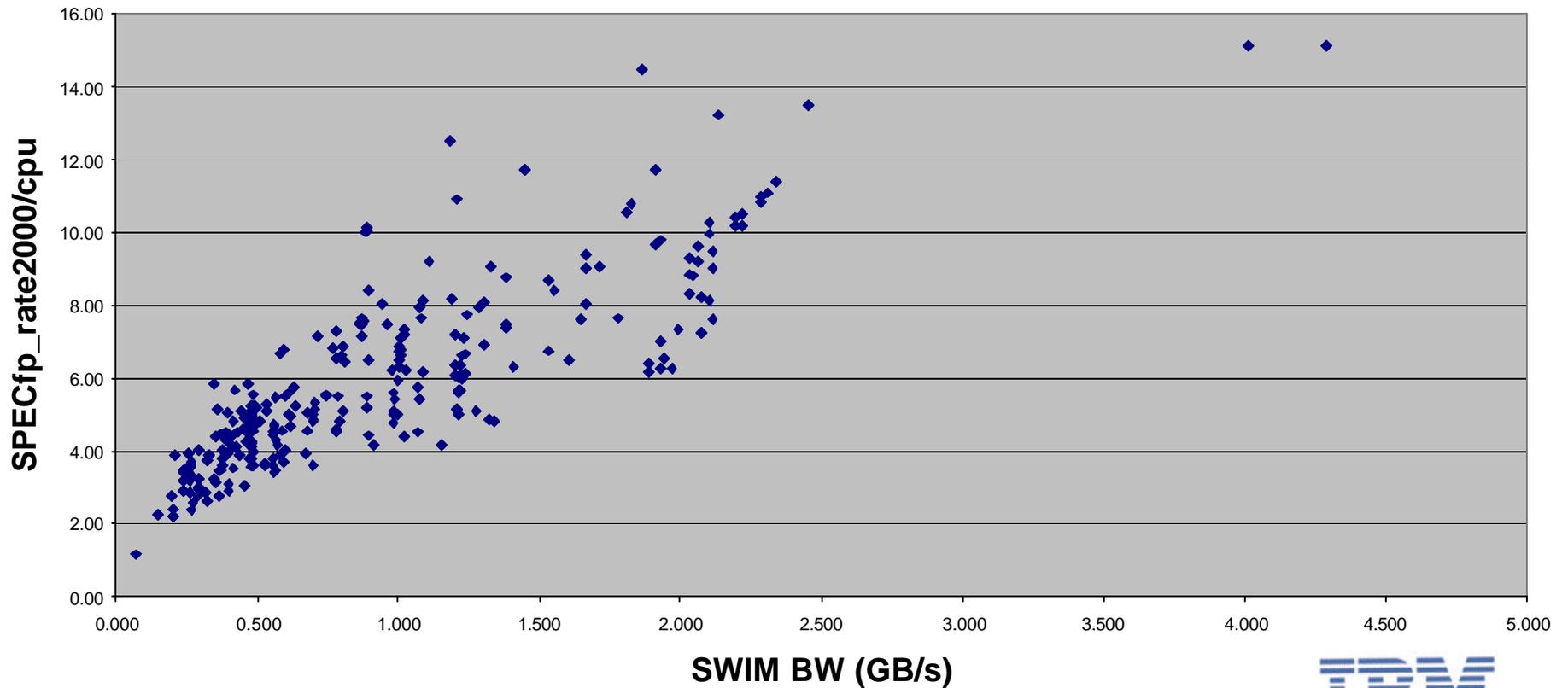
Does Peak GFLOPS predict SPECfp_rate2000?

SPECfp_rate2000/cpu vs Peak GFLOPS



Does Sustained Memory Bandwidth predict SPECfp_rate2000?

SPECfp_rate2000 vs SWIM BW



A Simple Composite Model

- Assume the time to solution is composed of a compute time proportional to peak GFLOPS plus a memory transfer time proportional to sustained memory bandwidth
- Assume “1 Byte/FLOP” to get:

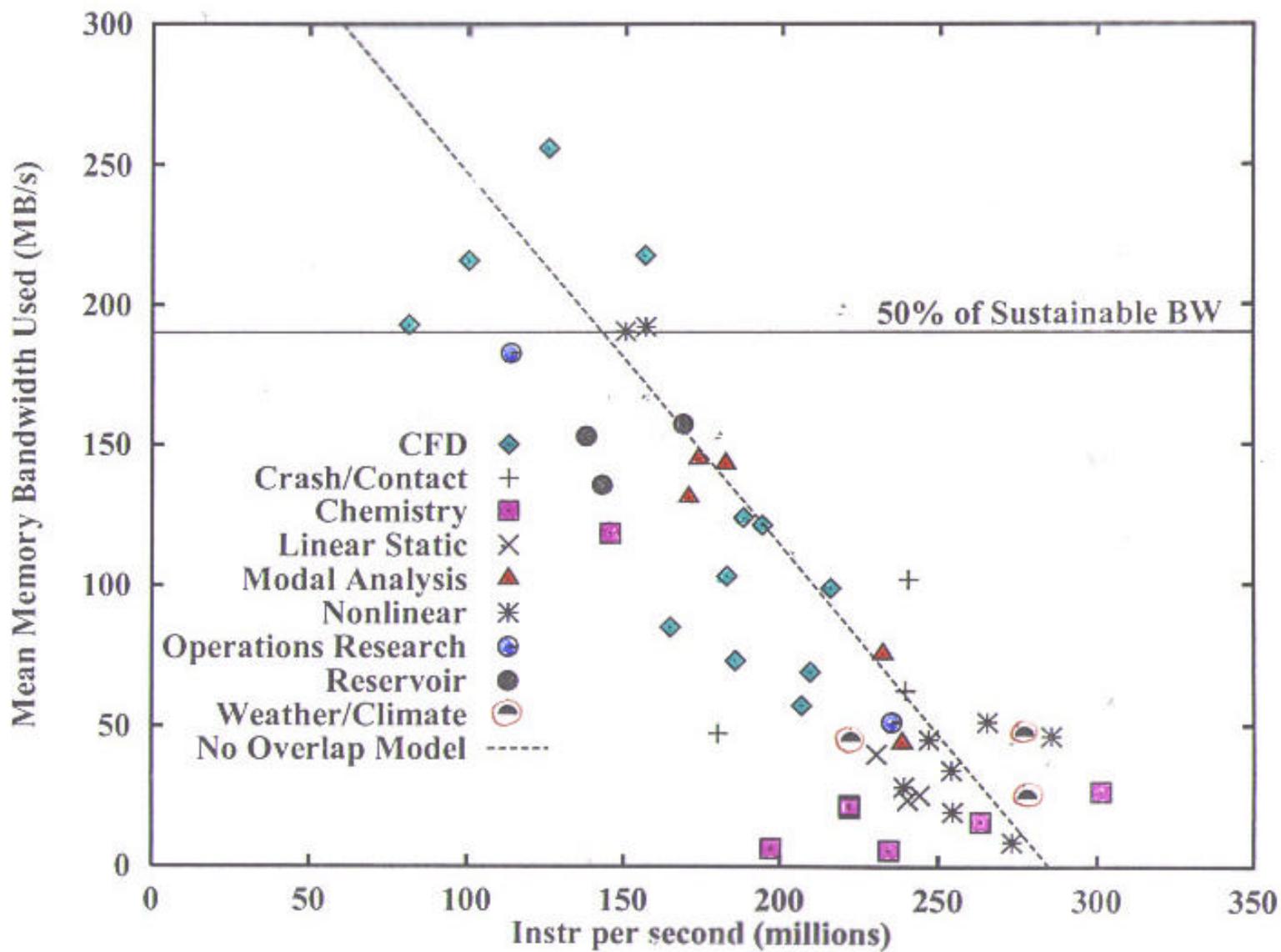
$$\text{"Effective GFLOPS" ?} = \frac{1 \text{ "Effective FP op"}}{\frac{?}{?} \frac{1 \text{ FP op}}{\text{? Peak GFLOPS}} + \frac{?}{?} \frac{1 \text{ Byte}}{\text{? Sustained GB/s}}}$$

- Use performance of 171.swim from SPECfp_rate2000 as a proxy for memory bandwidth

$$\text{Sustained BW} = (478.3 \text{ GB} * (\# \text{ of copies})) / (\text{run time for 171.swim})$$

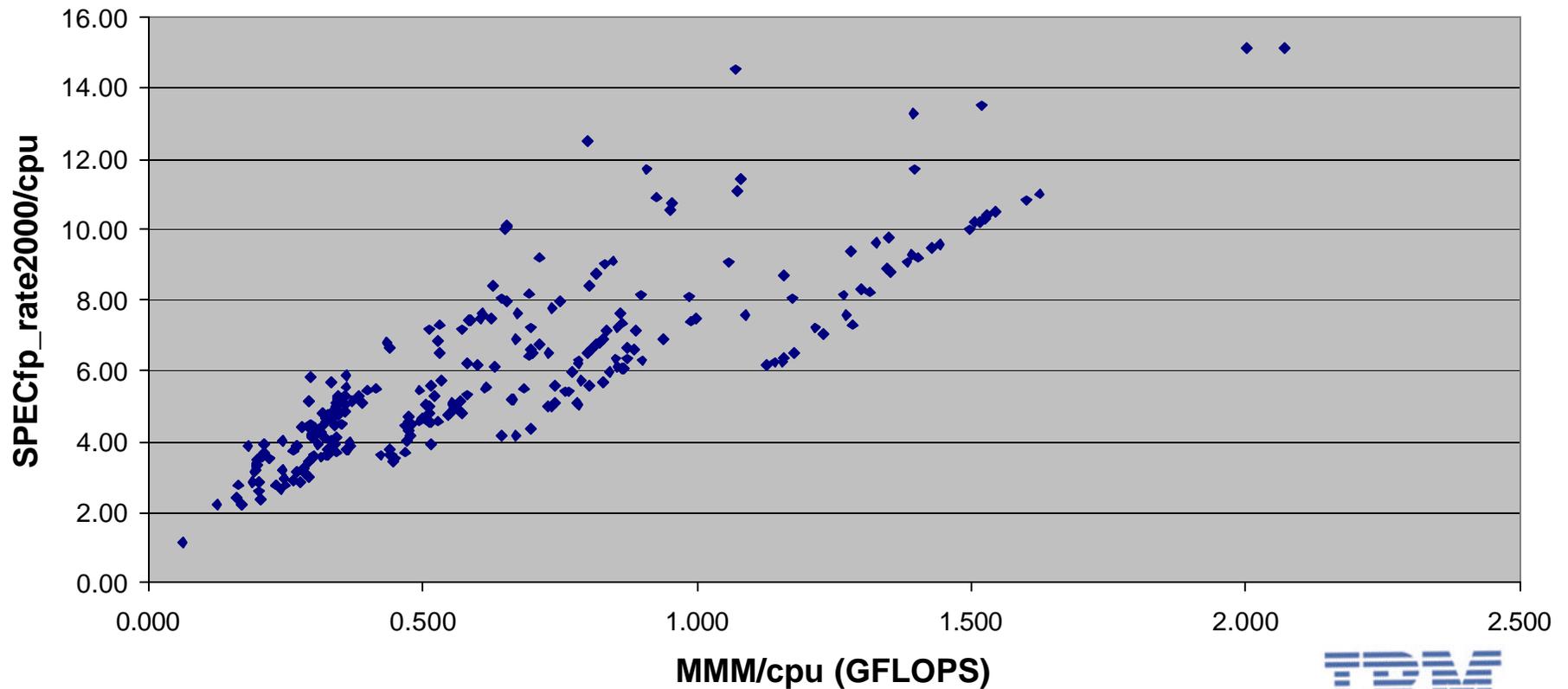


Why 1 Byte/FLOP ?



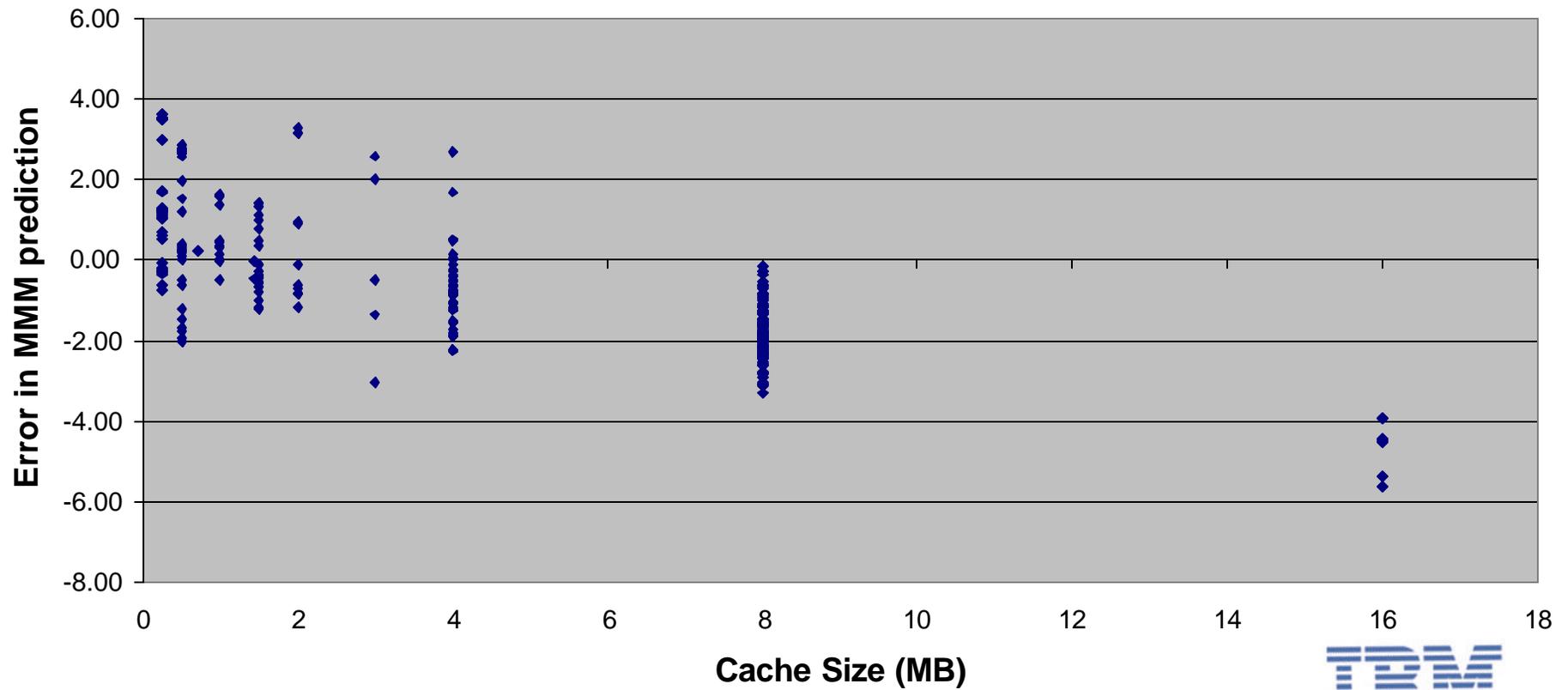
How does the “1 Byte per FLOP” model do?

SPECfp_rate2000/cpu vs MMM/cpu



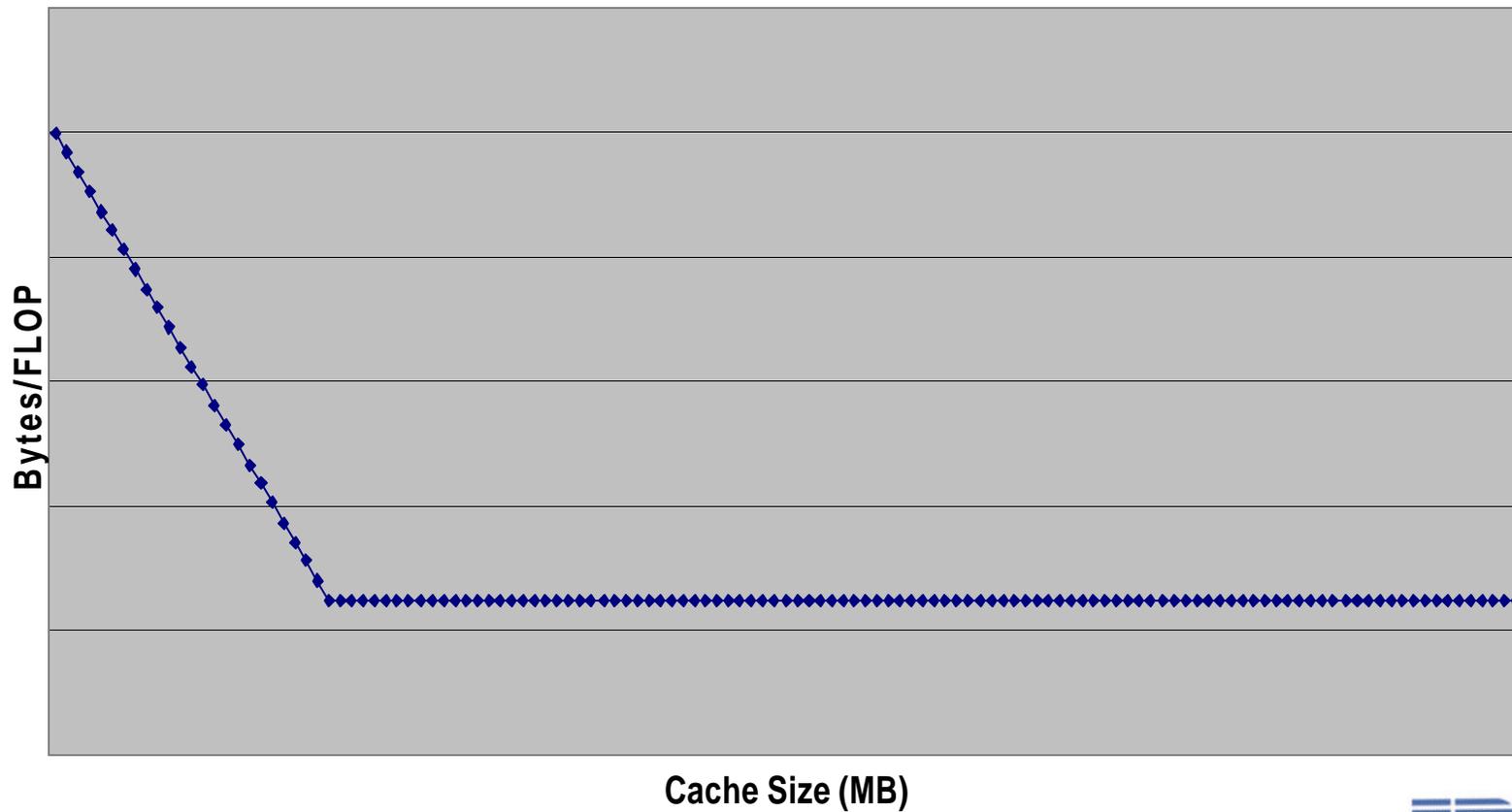
“1 Byte/FLOP” model error vs cache size

MMM error vs cache size



Make “Bytes/FLOP” a simple function of cache size

Assumed Bytes/FLOP



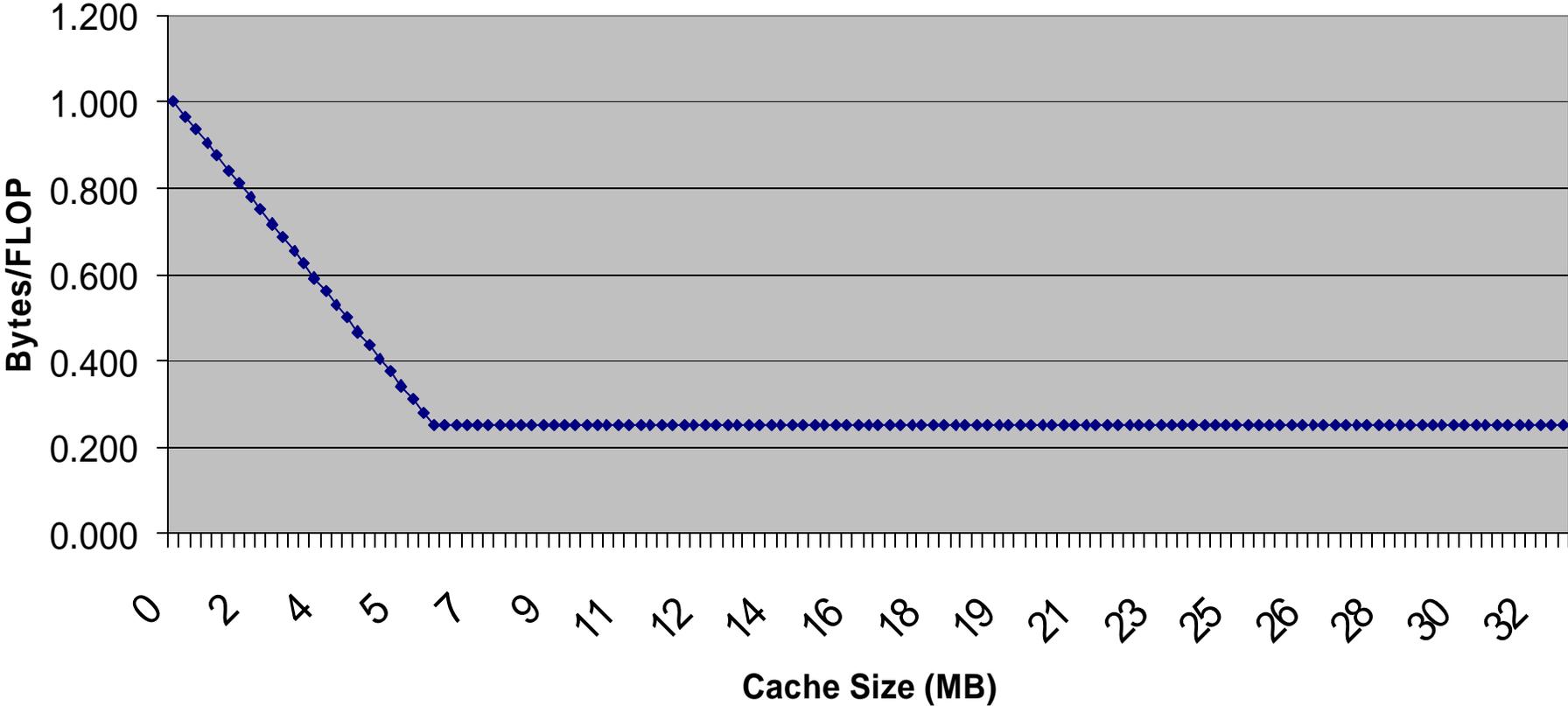
Make “Bytes/FLOP” a simple function of cache size

- Minimize RMS error to calculate the four parameters:
 - Bytes/FLOP for large caches
 - Bytes/FLOP for small caches
 - Size of asymptotically large cache
 - Coefficient of best-fit to SPECfp_rate2000/cpu
- Results (rounded to nearby round values):
 - Bytes/FLOP for large caches === 0.333 (*)
 - Bytes/FLOP for small caches === 1.00
 - Size of asymptotically large cache === 6 MB
 - Coefficient of best fit === 6.7 (+/- 0.1) (*)
 - The units of the coefficient are
SPECfp_rate2000 / Effective GFLOPS

(*) Revised to  2002-11-22

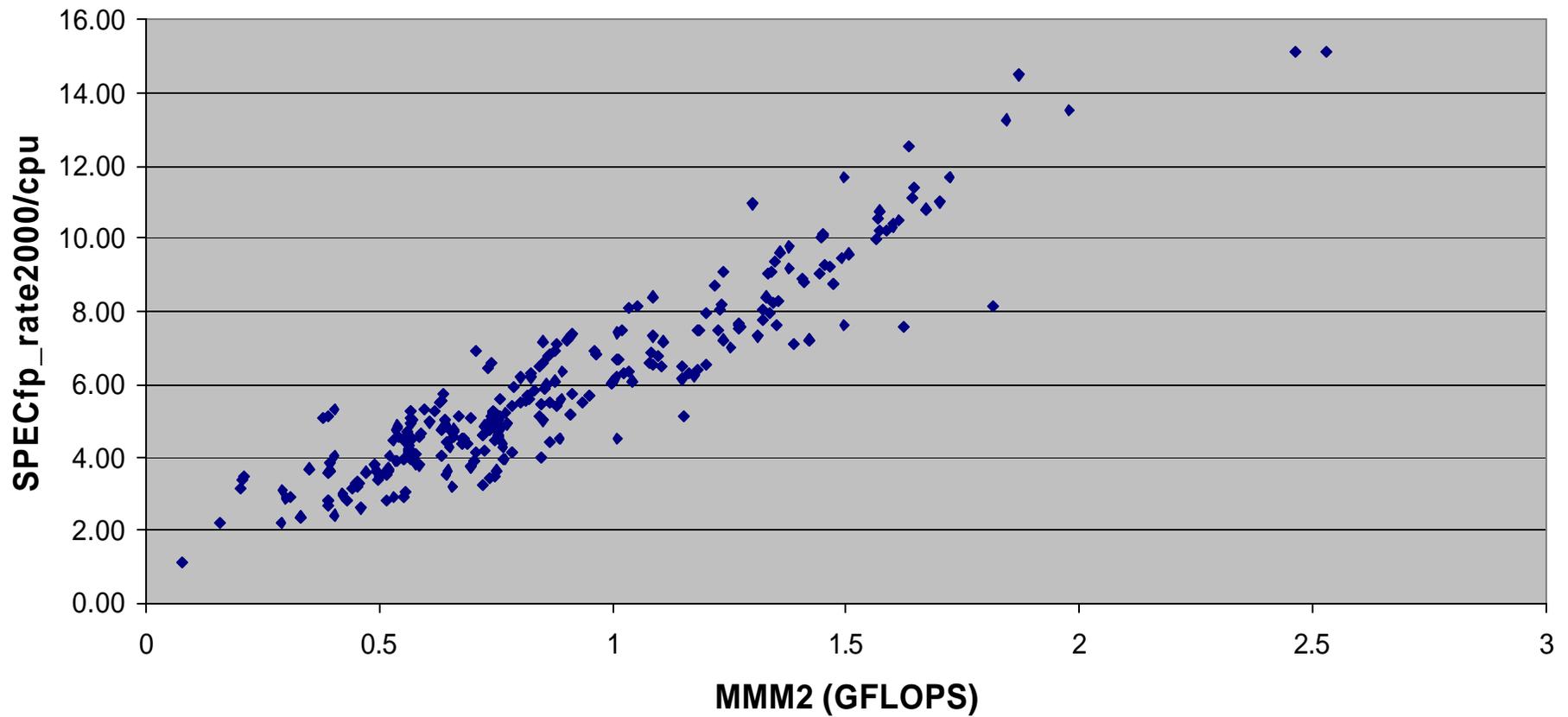
Make “Bytes/FLOP” a simple function of cache size

Assumed Bytes/FLOP

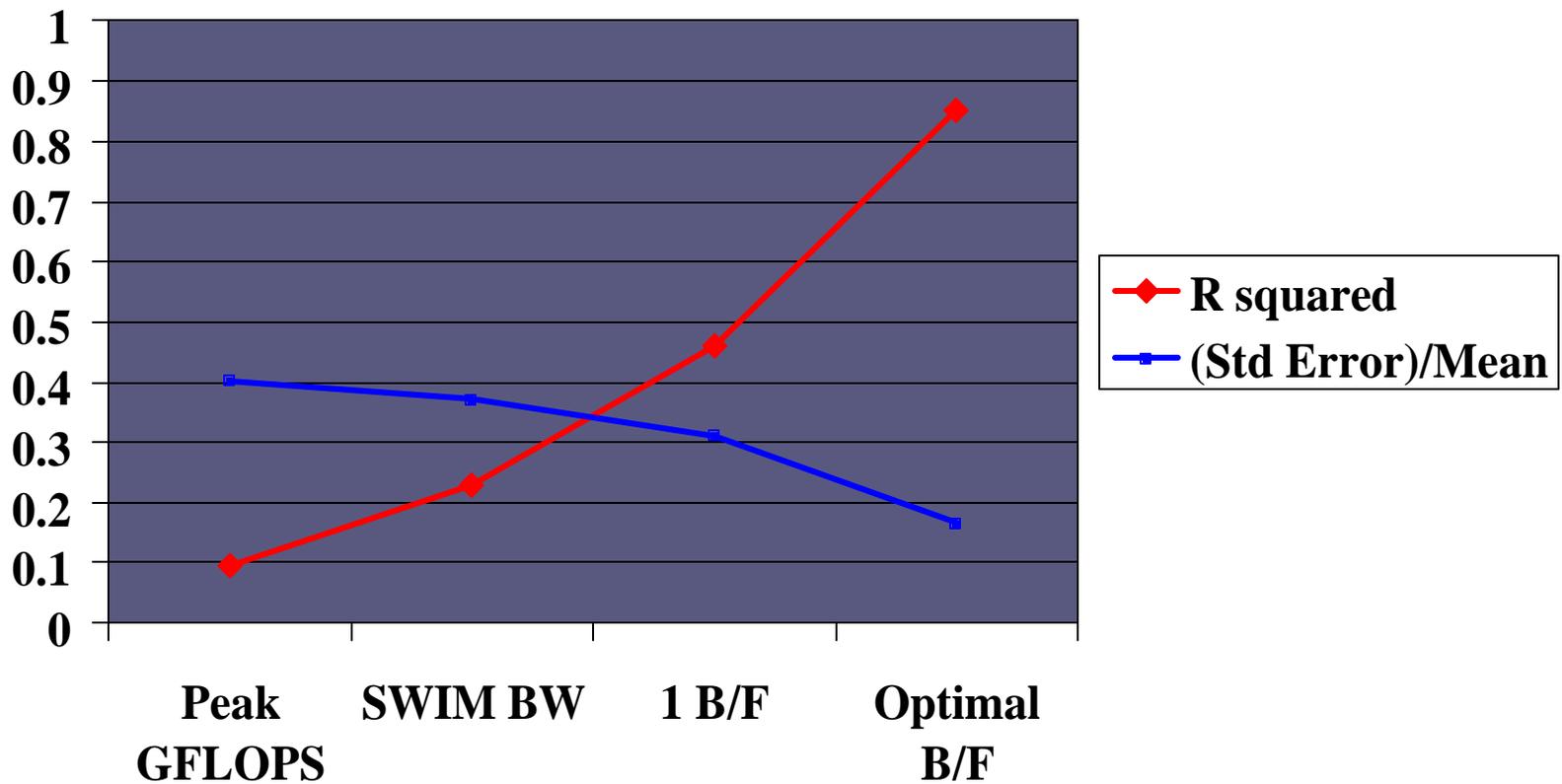


Does this Revised Metric predict SPECfp_rate2000?

SPECfp_rate2000/cpu vs MMM2

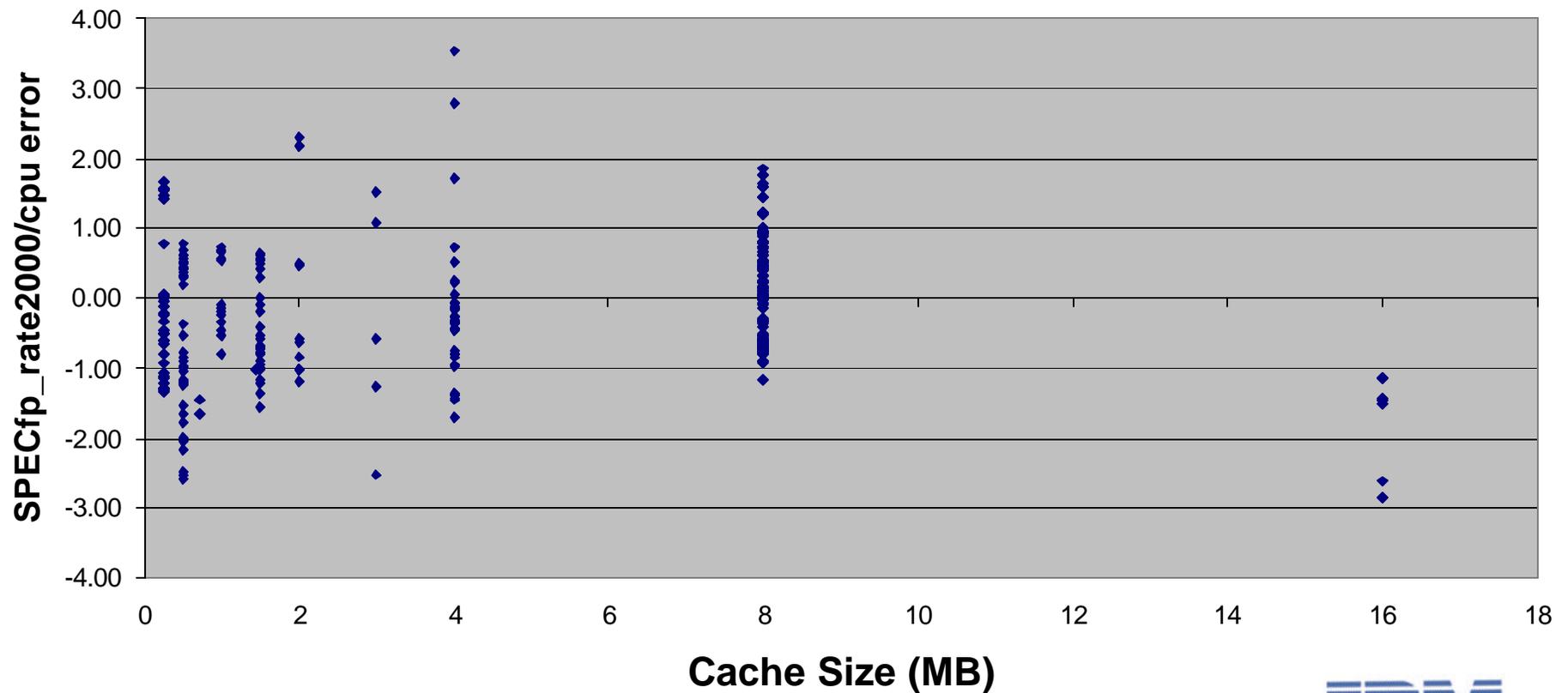


Statistical Metrics

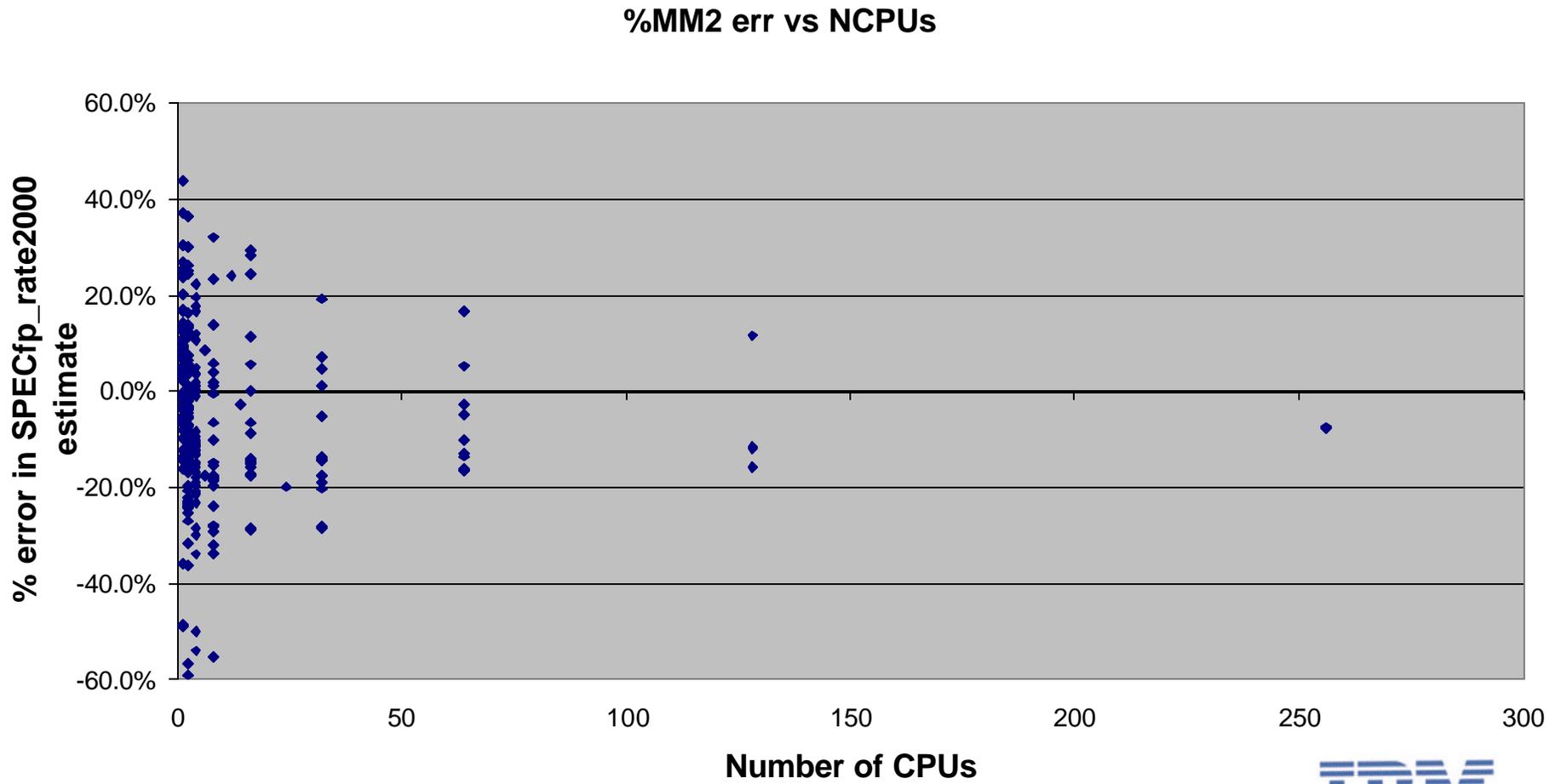


Does the Optimized Metric have Systematic Errors vs Cache Size?

MMM2 err vs \$ size



Does the Optimized Metric have Systematic Errors vs SMP size?



More Comments

- There is some evidence of a systematic error related to load/store capability vs FP capability
 - Systems with one load or store per peak FLOP (e.g., EV6, PentiumIII) do better on SPECfp_rate2000 than the optimized metric suggests
 - Systems capable of 4 FP ops per clock do less well on SPECfp_rate2000 than the metric suggests



More Comments (continued)

- Unfortunately, recasting in terms of peak LoadStore rate does not improve the statistics
- Arbitrary harmonic combinations of Peak GFLOPS and Peak LoadStore provide only a little improvement in the statistics, though it does eliminate a few of the outliers
 - R-squared increases from 0.85 to 0.87
 - Normalized Std Error decreases from 16% to 15%

Comments

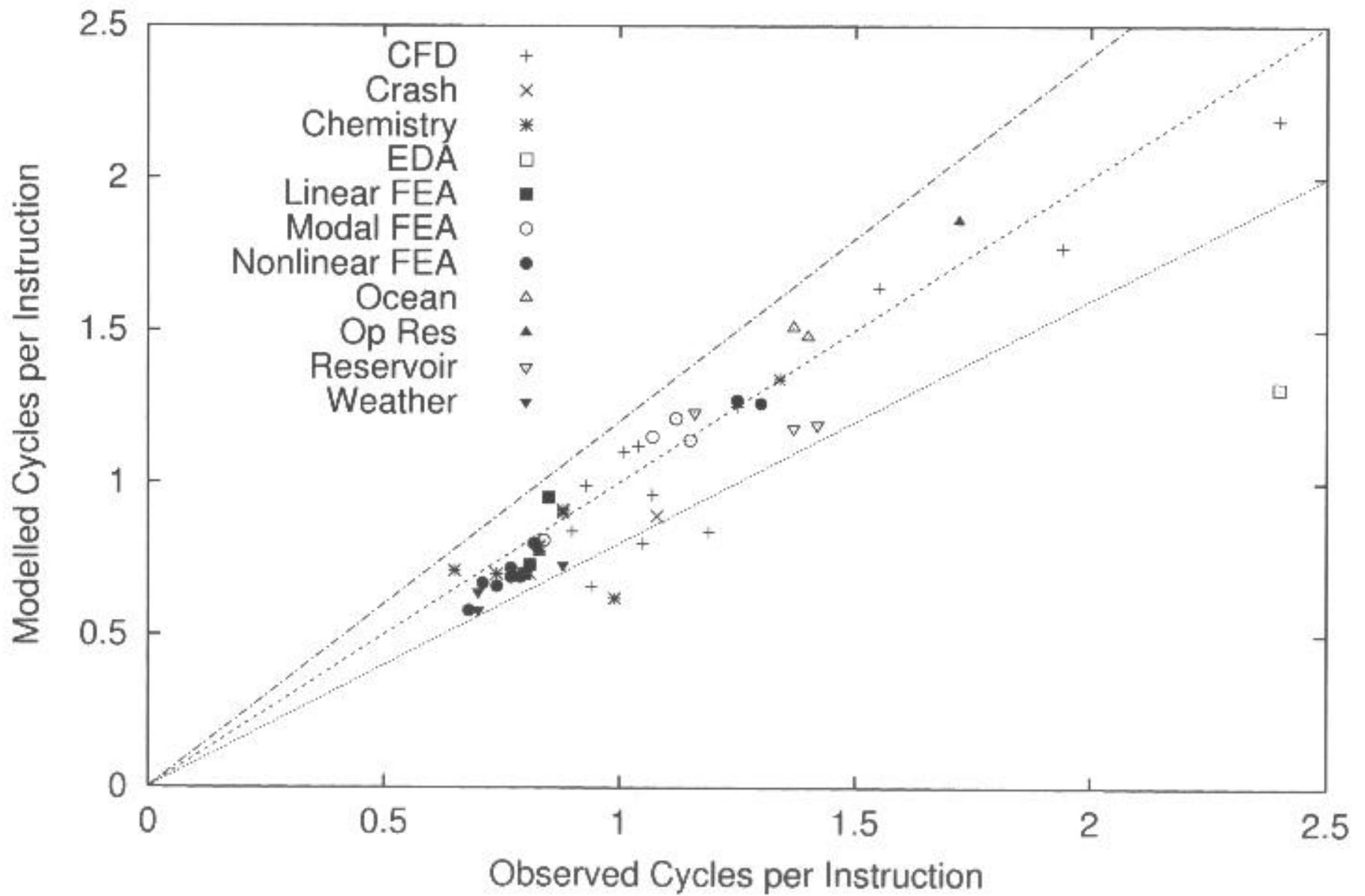
- Obviously, these coefficients were derived to match the SPECfp_rate2000 data set, not a “typical” set of supercomputing applications
- However, the results are encouraging, delivering a projection with 15% accuracy (one sigma) using a model based on only one measurement (sustainable memory bandwidth), plus specification of several architectural features

One More Attempt

- A more complex model (10 DOF) was created based on Origin2000 performance counters with a heuristic model for overlap
- The W_i 's were obtained from Origin2000 performance counters
- The goal was to see how well the heuristic model could reproduce the observed performance of the Origin2000 on a set of 25 ISV and community applications



Simple CPI Results for Applications



Comments on 10 DOF model

- About 75% of predictions were within 10% of observed results
- A few outliers were probably caused by excessive I/O (which was not measured by the performance monitoring facility)
- The model was very simple and was not tuned to fit the data
- Slightly better than the SPEC projections earlier, but the data sets were different, and the projections were all for single-threaded jobs running on a single system (the Origin2000)



Final Summary

- The composite methodology is
 - Simple to understand
 - Simple to measure (but aggregating W_i 's will be much work)
 - Based on a mathematically correct model of performance
 - Based on a backwards-looking view of performance bottlenecks
- As an example, adding STREAM benchmark data to the TOP500 database would allow such metrics to be constructed
- Obviously, much work remains on consistent modelling of the influence of non-local data transfer in the performance of supercomputers
 - The good news is that LogP models have been very successful in the past, using methodologies similar to what I have described here

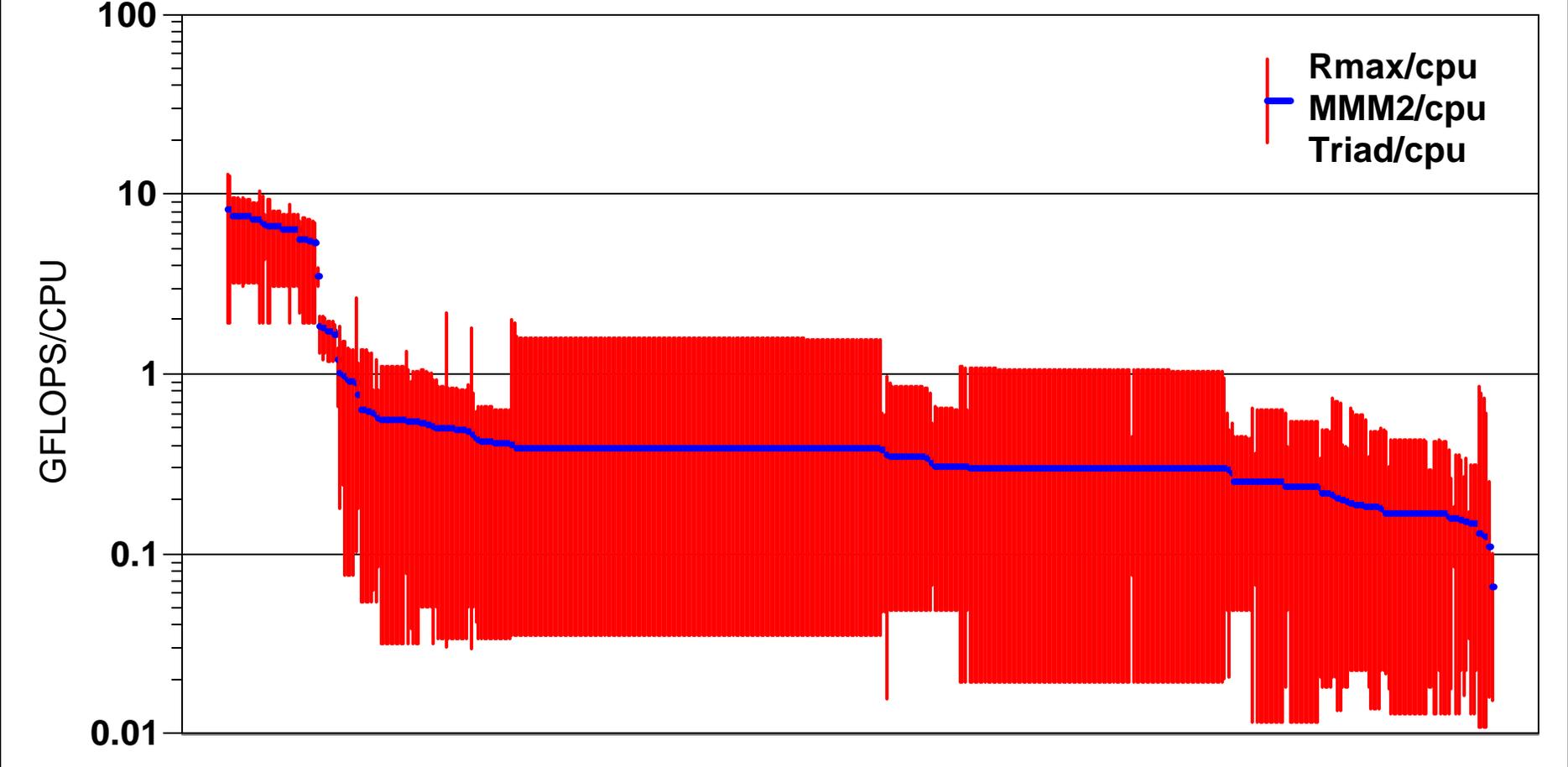


I lied, I am not really done yet

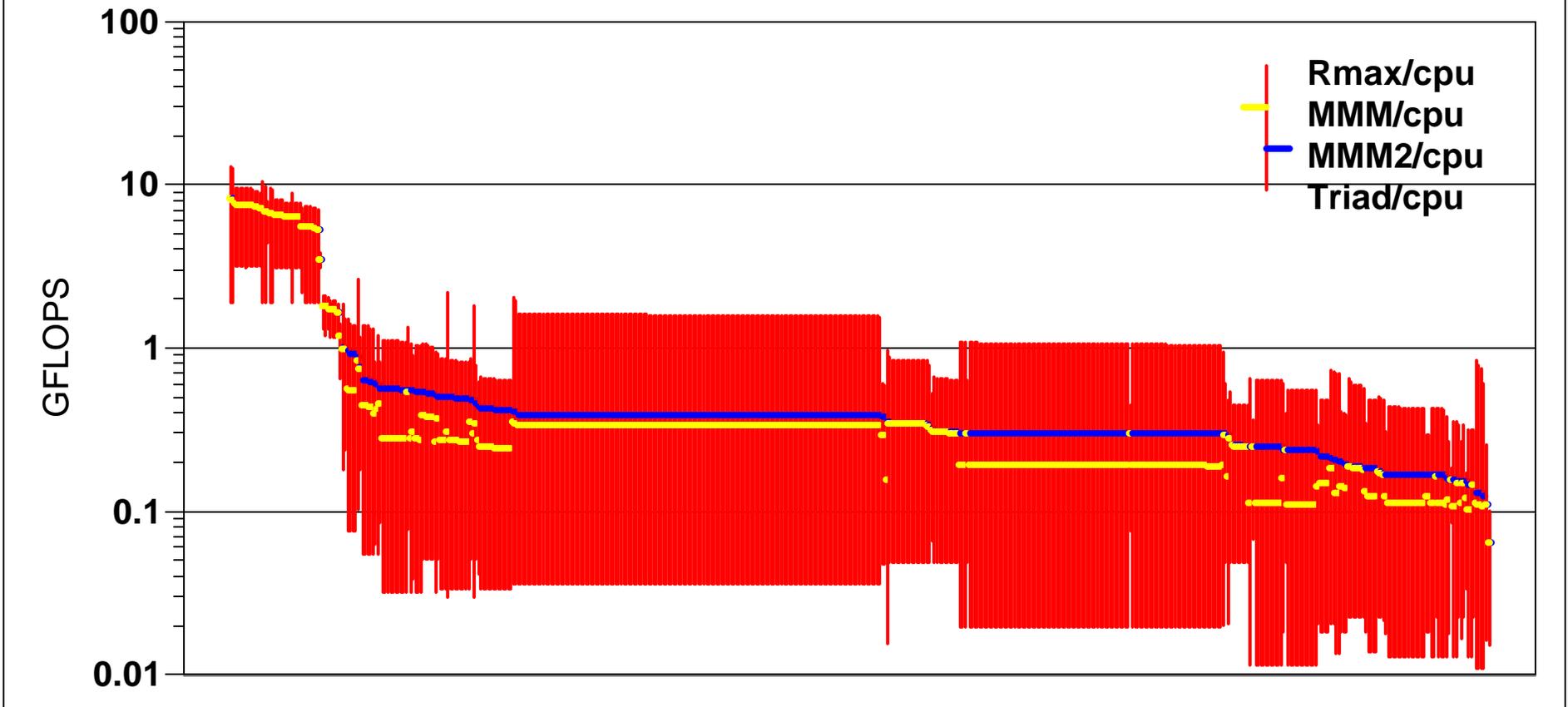
- I applied the MMM2 methodology to the TOP500 list
- I estimated cache sizes and STREAM Triad bandwidth for all 500 systems
- I used the same Bytes/FLOP parameter as in the SPECfp_rate2000 study
 - 1 B/F for small caches
 - 0.33 B/F for large caches
 - 6 MB is the cut-off for “large” caches



Performance Ranges per CPU



Performance Ranges MMM vs MMM2



Comments

- Results shown per cpu
 - Earth Simulator is at position #30
- Sorted by MMM2 (cache-size-dependent MMM)
- Lower bound is STREAM Triad MFLOPS
 - Equal STREAM Triad MB/s divided by 12 Bytes/FLOP
- Upper bound is LINPACK Rmax
- Both MMM and MMM are rational and quantitative methodology for picking an intermediate value in this (very large) range, with different interpretations of the impact of cache size on bandwidth requirements

Backup Slides

Notes about methodology and about
the STREAM benchmark



Methodology

- I extracted all the SPECfp_rate2000 (peak) results from the SPEC database on about 2002-11-12
 - Include cpu type, cpu frequency, cache sizes
 - Include 171.swim performance (time in seconds)
 - The data set included 331 results
- I added “peak FP ops/Hz” based on other published sources
 - Revisions on 2002-11-21 corrected this value for Fujitsu systems
- I defined “cache” as the largest on-chip cache or off-chip SRAM cache.
 - Off-chip DRAM caches are much less effective at improving performance, and degrade the accuracy of the proposed metric

The STREAM benchmark

- Not as old as LINPACK, but older than the TOP500 list
 - First published in 1991, revised as new results delivered
 - Currently contains 750 results in the table
 - Covers a very wide variety of systems
- STREAM is also an accidental benchmark
 - Developed in 1990-1991 to show why available systems had such wide variation in performance on my ocean models



STREAM (continued)

- STREAM is very simple
 - Measure the sustained memory bandwidth for a set of four computational kernels
 - COPY: $A(I) = B(I)$
 - SCALE: $A(I) = Q * B(I)$
 - ADD: $A(I) = A(I) + B(I)$
 - TRIAD: $A(I) = A(I) + Q * B(I)$
 - Make sure that each of the vectors is chosen to be larger than the available cache(s)

STREAM (continued)

- STREAM is not intended to predict performance
- STREAM acts as a counterpoint to LINPACK
 - LINPACK does not require significant memory bandwidth
 - STREAM does not require significant computational capability
- Can LINPACK and STREAM be used together to provide much more information to the public than either metric in isolation?