

System Performance "Balance",
System Cost "Balance",
Application "Balance"
& the SPEC CPU2000/CPU2006 Benchmarks

John D. McCalpin, Ph.D.
Principal Scientist, AMD

Presented by Joerg Borowski, AMD

Today's Talk

- "System Balance", "Application Balance", and "Optimization" are commonly used terms, but are typically not clearly defined
 - Definitions
 - Examples
 - Performance Models
- "Price/Performance" is a popular metric for system optimization
 - Define cost (price) model
 - Combine with previous performance models
 - Discuss non-intuitive results
- Application to the SPEC CPU2000 FP and SPEC CPU2006 FP benchmarks
 - Theoretical price/performance optimization
 - "Real World" price/performance optimization



What are “System Balance” and “Application Balance”?

- Webster’s Online Dictionary has 9 definitions of “balance” as a noun. The two interesting ones are:
 - **5 a** : stability produced by even distribution of weight on each side of the vertical axis **b** : equipoise between contrasting, opposing, or interacting elements [*quantitative, but limited to one dimension*]
 - **6 a** : an aesthetically pleasing integration of elements [*not quantitative*]
- So when we talk about computer “balance” or application “balance”, do we know what we are talking about? [*Usually not*]
- Creating your own definitions of “balance” can be educational
 - How many variables? Which ones? How are they weighted?
 - Performance: CPU, cache, DRAM capacity, DRAM bandwidth, I/O?
 - Physical Attributes: Size, Power, Expandability, Serviceability?
 - Cost: Purchase, Support, Power/Cooling?
- Today’s talk focuses on a typical definition for “System Balance” and “Application Balance”
 - Balance = (Bytes/second of memory BW) / (CPU Cycles of computation)



What is “Optimization”

- Another very tricky word...
- Webster’s online:
 - : an act, process, or methodology of making something (as a design, system, or decision) as fully perfect, functional, or effective as possible; *specifically* : the mathematical procedures (as finding the maximum of a function) involved in this
- “Optimization” does not mean making something “better” in the abstract – it means maximizing or minimizing a specific objective function
- What is **your** objective function?
 - Optimize [single-thread, multi-thread, throughput] x [performance, performance/cost, performance/watt]
 - Satisfy constraints: [max hardware cost, max physical size, minimum memory, minimum I/O, expandability, max allowable time-to-solution, software compatibility]
 - I count at least 20 primary purchasing metrics in common use for servers



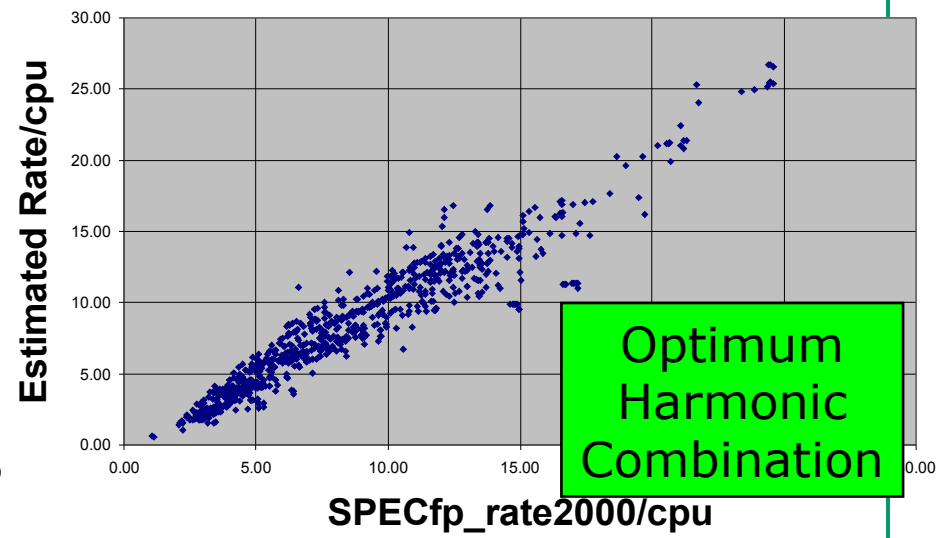
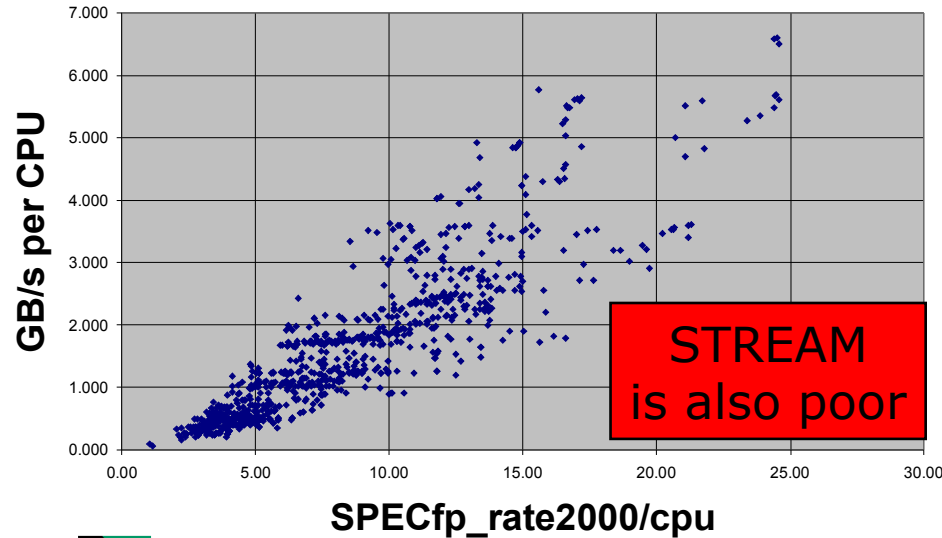
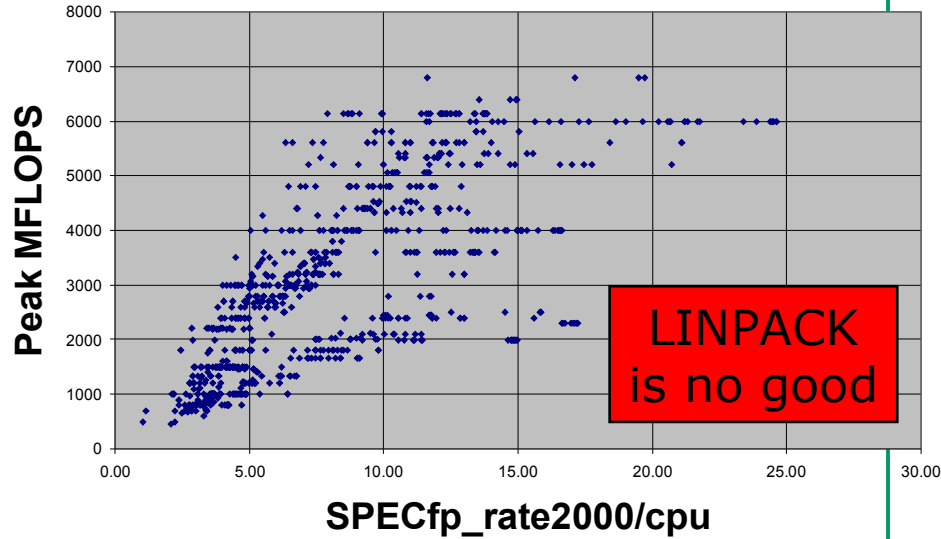
“Optimization” (continued)

- Computing is a human activity that can be modeled as a rational economic decision
- The fundamental metrics are therefore
 - **Value** – what *benefits* do I obtain from doing this computing?
 - **Cost** – what do I have to *give up* to do this computing?
- These may be only loosely related to technology metrics like “performance” or “price”
 - E.g., “Value” is typically bounded as time-to-solution goes to zero – i.e., as “performance” goes to infinity
 - E.g., “Value” is typically bounded as “price” goes to zero – how many computers can I use, even if they are “free”?
- I can’t define your “Value” or “Cost” metrics
 - So my examples will be based on “Performance” and “Price”
 - This does not negate the critical importance of the differences between “Value” and “Performance” & “Cost” and “Price” !



A Simple Performance Model

Consider the task of projecting
SPECfp_rate2000 using as few
parameters and/or
measurements as possible
“McCalpin’s Magical Metric” obtains
~16% RMS error using 1
measurement plus 2
architectural parameters



Slightly Less Simple Performance Models

- “McCalpin’s Magical Metric” can be made slightly more accurate by developing separate models for each of the 14 benchmarks and then taking the geometric mean of the results
 - It is very hard to get to 10% accuracy without adding many more degrees of freedom to the model or adding significantly more parameters to describe microarchitectures and/or software
- Dramatically more accurate models can be made for systems based on a single processor family and fixed compiler choice
- Here I consider models for SPEC CFP2000 and CFP2006 (speed and rate) for systems using AMD Opteron processors
 - All systems have the same CPU microarchitecture and same cache sizes
 - All systems are running “similar” versions of Linux and effectively identical compilers (PathScale EKOPath 2.5 compiler suite)
 - Each of the 14 SPECfp2000 and 17 SPECfp2006 codes has its own model
- Two parameter models are not flexible enough to get <10% errors
 - Adding a “Memory Latency” component reduces errors to <<5%



Slightly Less Simple Model, continued

- The model is additive (ignores overlap) and is of the form:

$$T = \frac{W_{CPU}}{R_{CPU}} + \frac{W_{BW}}{R_{BW}} + \frac{W_{latency}}{R_{latency}}$$

- The three "Rate" parameters are specified *a priori* based on the system configuration:

Metric	Description	Value on Reference System ⁽¹⁾
R_{CPU}	CPU GHz	2.8 GHz
R_{BW}	Sustained GB/s per SPEC "user"	2.6 GB/s per "user"
$R_{latency}$	Idle pointer-chasing memory latency of system	60 ns

(1) Reference Hardware Configuration: Two socket server with two 2.8 GHz dual-core AMD 2000 series processors & four 667 MHz DDR2 DIMMs per socket



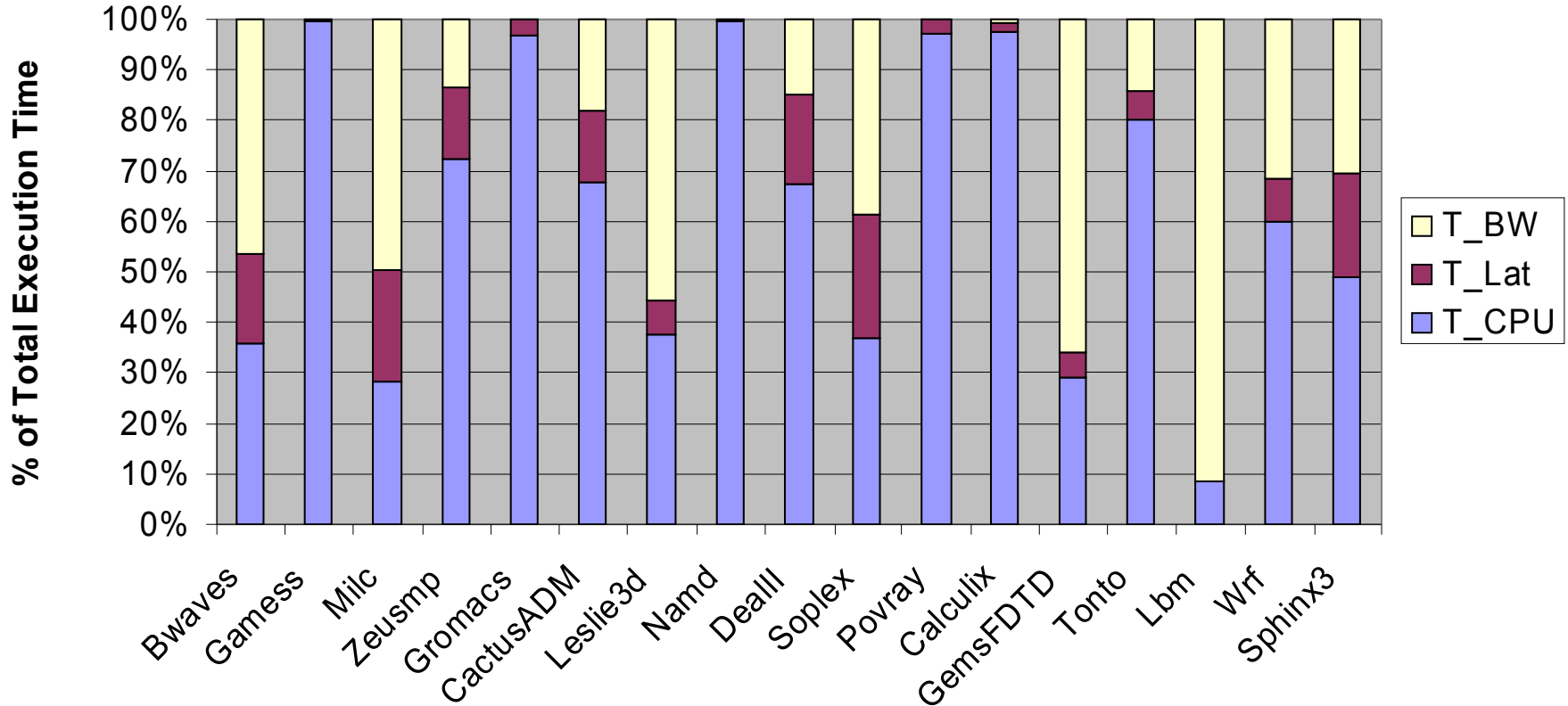
Slightly Less Simple Model, continued again

- The three “Work” components are based on a least-squares fit of the published execution time to the model projection for each benchmark
 - Separate models for each benchmark (14 CFP2000 + 17 CFP2006)
- Published Data
 - CPU2000: 9 “speed”, 20 “rate”, 6 independent system configurations
 - CPU2006: 7 “speed”, 22 “rate”, 13 independent system configurations
- The models are strongly consistent with the measured results:

RMS error of projected execution time vs published results	CFP2006 17 benchmarks	CFP2000 14 benchmarks
4% - 6%	2 (12%)	0 (0%)
2% - 4%	7 (41%)	4 (29%)
0% - 2%	8 (47%)	10 (71%)

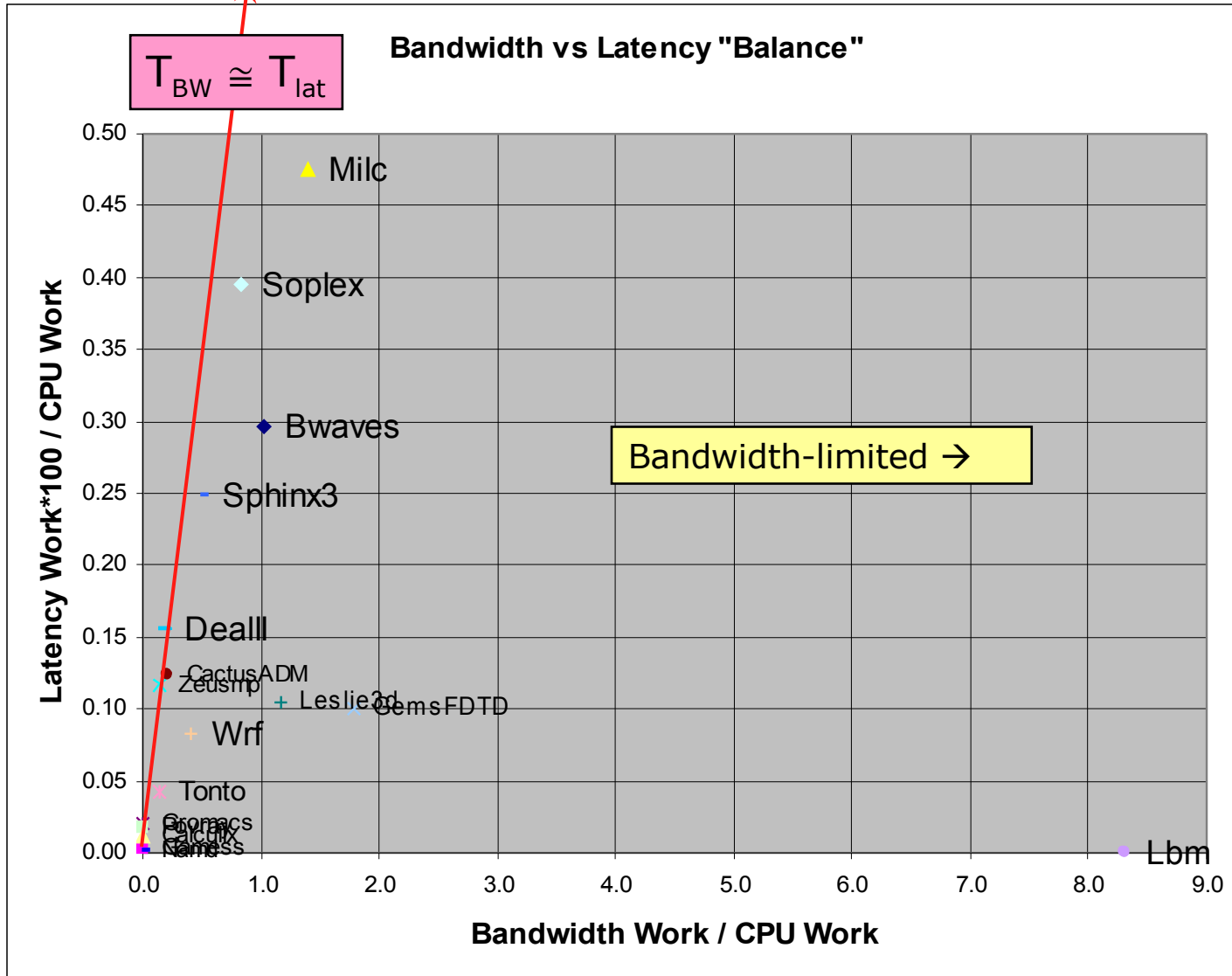
CFP2006 Execution Time on Reference System

Modelled Execution Time Breakdown for Reference System



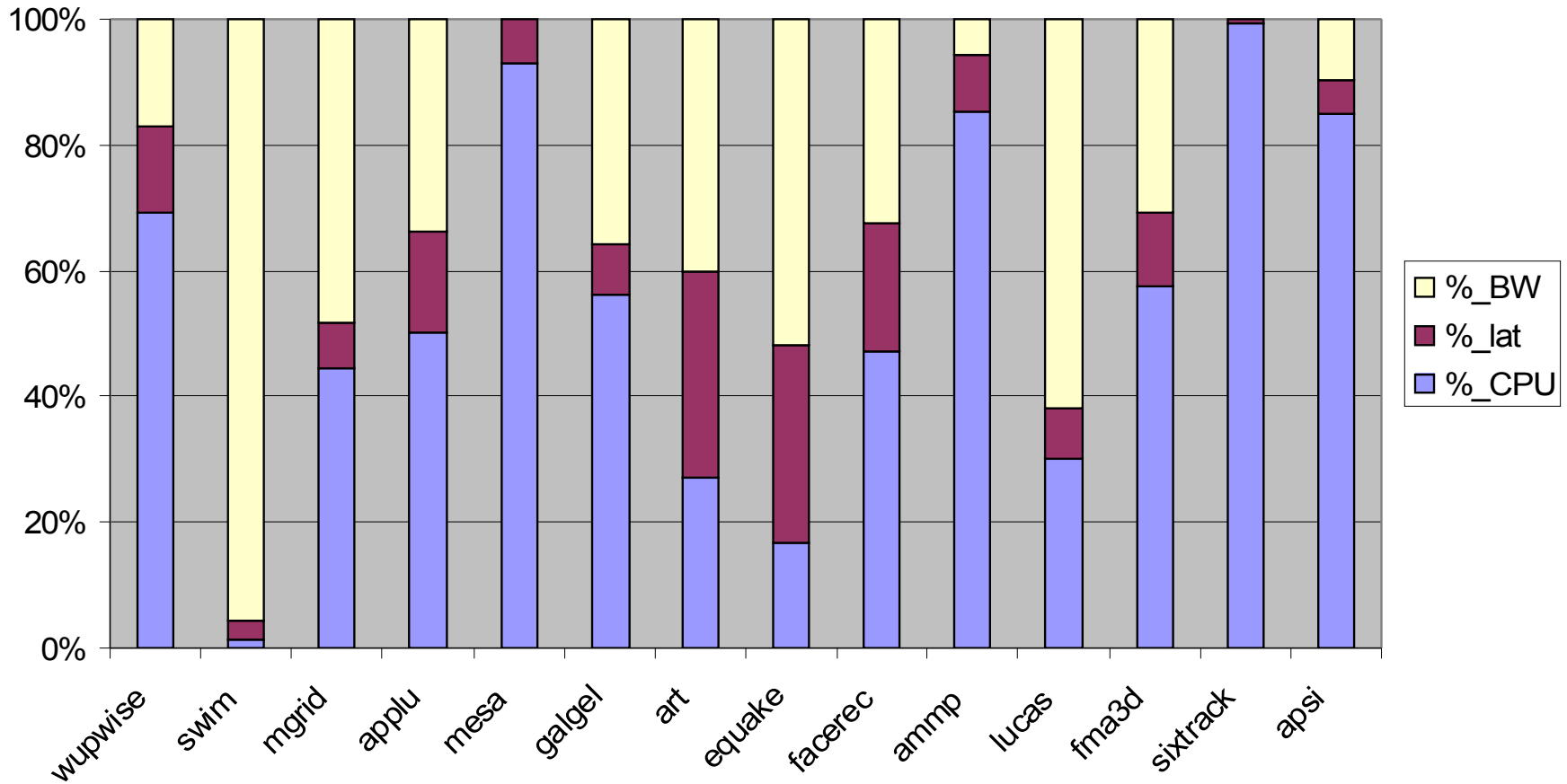
Latency and Bandwidth "Balance" vs CPU work

Latency-limited →



CFP2000 Execution Time on Reference System

Time Breakdown for CFP2000



Performance Model Summary

- Simple performance models with a few additive components can be fit very accurately to each of the SPEC CFP2000_rate and SPEC CFP2006_rate benchmarks
- The SPEC CFP2000 and CFP2006 benchmarks both show a very broad range of “Application Balance”, ranging from CPU-limited to (almost) Memory Bandwidth-limited, with a substantial subset showing non-trivial Memory Latency stall times.
- The broad range of these results and the accuracy of the models suggest that the models will be quite useful in further modeling of Price/Performance



Cost Model

- In principal, additive cost models can be constructed for computer systems
- Difficulties & Limitations
 - Costs can vary dramatically due to features that do not impact performance (e.g., reliability, scalability, low power, etc)
 - Many theoretically possible configurations are not available because no one has chosen to build them
- Key Feature: Cost models have a large “baseline” component associated with the “stuff” necessary to support the CPU and memory (motherboards, disk, power supplies, etc)
- For the initial analytical modeling, I will assume:

$$C = C_{base} + \$_{CPU} R_{CPU} + \$_{BW} R_{BW}$$

- C = cost, R = rate
- $\$_{CPU}$ = cost per unit CPU performance
- $\$_{BW}$ = cost per unit sustainable memory BW



Cost/Performance

- Cost/Performance \approx Cost * Time

$$C = C_{base} + \$_{CPU} R_{CPU} + \$_{BW} R_{BW} \quad \times \quad T = \frac{W_{CPU}}{R_{CPU}} + \frac{W_{BW}}{R_{BW}} + \frac{W_{latency}}{R_{latency}}$$

- or

$$C = C_{base} + C_{CPU} + C_{BW} \quad \times \quad T = T_{CPU} + T_{BW} + T_{latency}$$

- Note that Cost * Time has 9 terms!
 - Two terms correspond to "active" useful work: $T_{cpu} * C_{cpu}$ and $T_{bw} * C_{bw}$
 - The remaining 7 terms are overheads:
 - Depreciation of active components while other active components are busy
 $T_{cpu} * C_{bw}$, $T_{bw} * C_{cpu}$
 - Depreciation of inactive components while the active components are busy
 $T_{cpu} * C_{base}$, $T_{bw} * C_{base}$
 - Depreciation of everything while waiting on memory latency stalls
 $T_{latency} * C_{base}$, $T_{latency} * C_{cpu}$, $T_{latency} * C_{bw}$



Optimization of Cost/Performance

- Define some nondimensional parameters:

$$\beta = R_{bw}/R_{cpu} \quad \leftarrow \text{machine balance}$$

$$\gamma = W_{bw}/W_{cpu} \quad \leftarrow \text{application balance}$$

$$\delta = \$_{bw}/\$_{cpu} \quad \leftarrow \text{technology cost balance}$$

- In the absence of the constant terms (C_{base} & $T_{latency}$), Cost*Time is a simple quadratic function with a simple analytic optimum

$$\beta_{optimum} = \sqrt{\frac{\gamma}{\delta}}$$

- In the presence of the constant terms the equation is effectively quadratic and the one physically realizable optimum provides specific values of R_{cpu} and R_{bw}

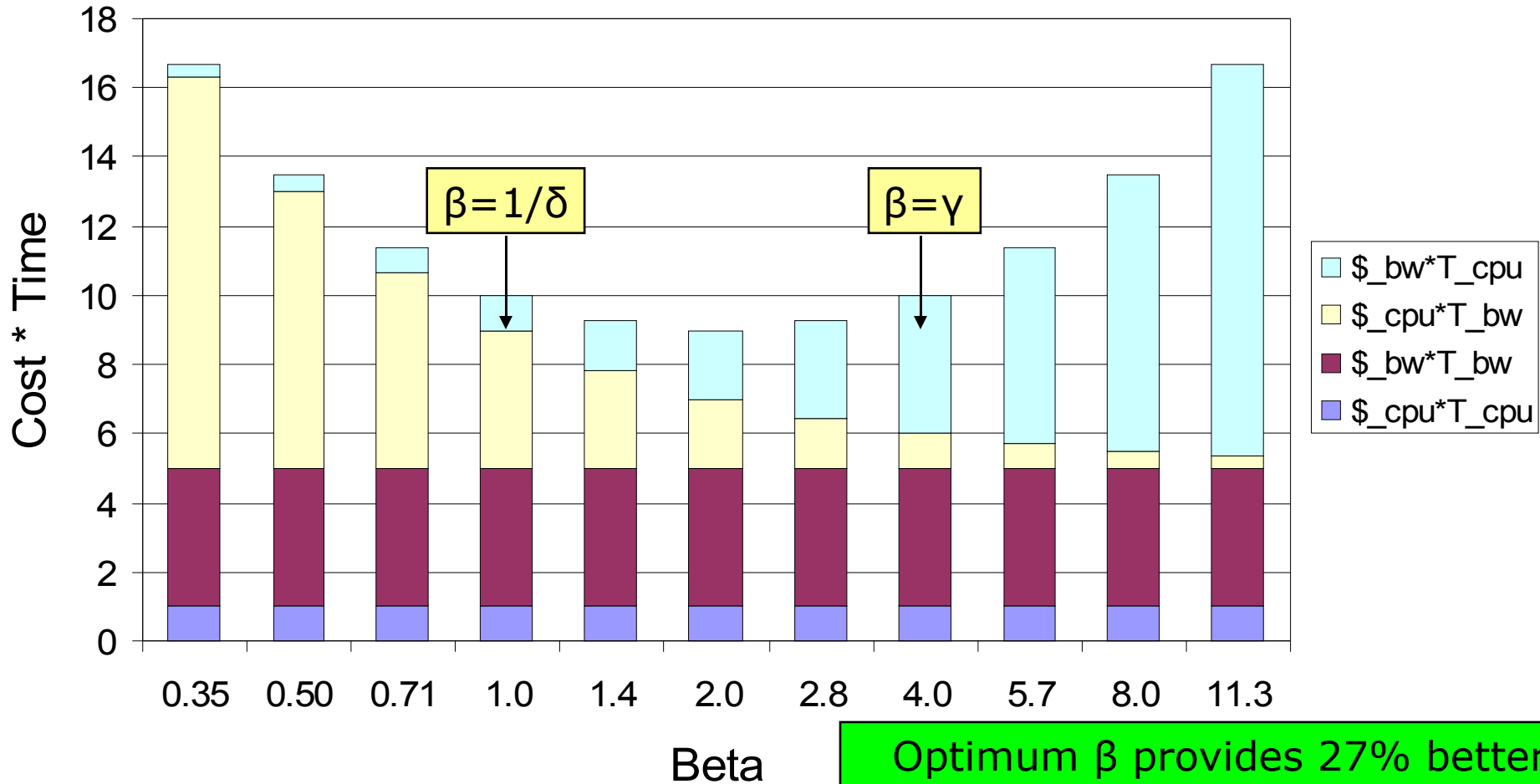
$$R_{CPU} = \sqrt{\frac{W_{CPU}}{\$_{CPU}}} \times \sqrt{\frac{C_{base}}{T_{latency}}}$$

$$R_{BW} = \sqrt{\frac{W_{BW}}{\$_{BW}}} \times \sqrt{\frac{C_{base}}{T_{latency}}}$$

Interestingly, the optimum machine balance ratio β is the same in both cases

A Simple Analytical Example (no constant terms)

Cost*Time Details: $\gamma=4$, $\delta=1$



Optimum β provides 27% better Cost/Performance than equal partitioning of time ($\beta=\gamma$) or equal partitioning of cost ($\beta=1/\delta$)

More detailed examples

- Given that we do have computers, we do not need to limit ourselves to analytical solutions.
- I obtained pricing for a wide variety of 1U, 2-socket Opteron servers from www.serversdirect.com during June 2007
 - Opteron “RevE” single-core CPUs @ 5 frequencies, with DDR/400 memory
 - Opteron “RevE” dual-core CPUs @ 6 frequencies, with DDR/400 memory
 - Opteron “RevF” dual-core CPUs @ 6 frequencies, with DDR2/667 memory
- The pricing for the servers was split into “base”, “CPU”, and “Memory” components
 - Yes, I am deliberately confusing “Memory Cost” and “Bandwidth Cost”, but I don’t have time to explain why this is less wrong than it might appear
 - Each system was configured with the cheapest option providing 2 GB/core
- Cost*Time was computed for each of the 31 benchmarks on each of the systems, and the configuration with the lowest Cost*Time was identified



Optimum Price/Performance for Opteron "RevE"



CPU/Memory	Cores/Chip	CPU GHz	Optimum for this many of the 14 CFP2000 benchmarks	Optimum for this many of the 17 CFP2006 benchmarks
Opteron RevE, DDR/400	1	2.2	1	-
Opteron RevE, DDR/400	1	2.4	-	1
Opteron RevE, DDR/400	2	1.8	2	2
Opteron RevE, DDR/400	2	2.0	2	3
Opteron RevE, DDR/400	2	2.2	9	11



Optimum Price/Performance for Opteron "RevF"

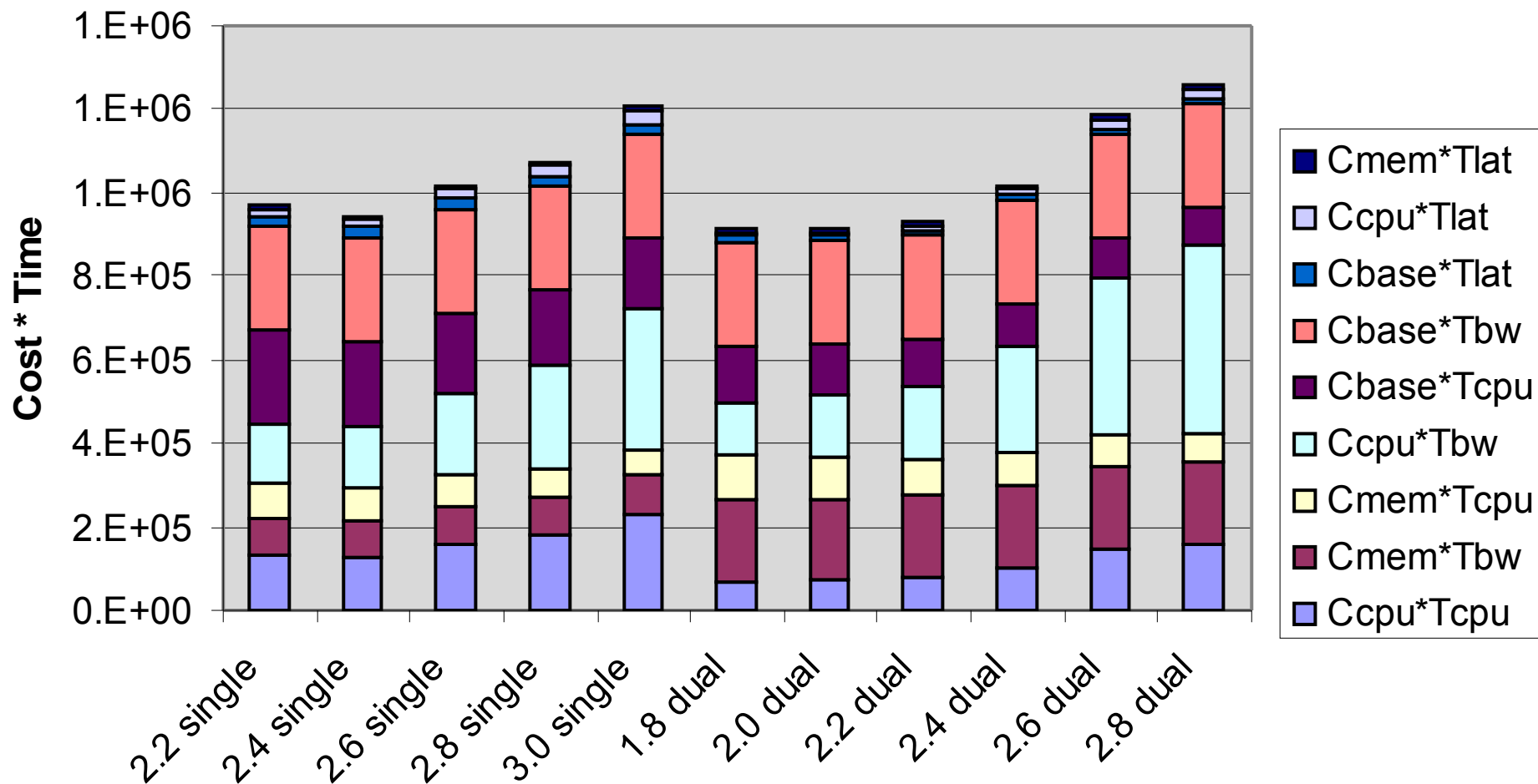


CPU/Memory	Cores/Chip	CPU GHz	Optimum for this many of the 14 CFP2000 benchmarks	Optimum for this many of the 17 CFP2006 benchmarks
Opteron RevF, DDR2/667	2	1.8	3	3
Opteron RevF, DDR2/667	2	2.0	3	4
Opteron RevF, DDR2/667	2	2.2	8	10

- Note that the Performance model can provide very accurate performance projections for both single-core and dual-core parts, but...
- AMD is not currently offering single-core RevF Opteron processors, so pricing is only available for the dual-core processors here



Detailed Cost*Time Breakdown for GemsFDTD



Failure and Success with Analytical Modeling

- Linear trends were fit to the CPU and memory cost data to create an analytically tractable cost model
- Optimum R_{cpu} and R_{bw} were computed for each of the 17 CFP2000 benchmarks
 - For 8 benchmarks the computed optimal R_{cpu} and R_{bw} were within physically realizable limits for systems based on the Opteron processor
 - For 3 benchmarks the computed optimal R_{bw} was much higher than is currently configurable
 - These benchmarks might benefit from single-core processors
 - For 6 benchmarks the computed optimal R_{cpu} was much higher than is currently configurable
 - These benchmarks should benefit from quad-core processors
- Bad News: poor numerical correlation between analytical and observed optimum system balance
- Good News: perfect correlation of rank ordering of analytical and observed optimum system balance



Benchmark	Analytic Optimum Beta	Observed Optimum Beta
Lbm	4.03	1.45
GemsFDTD	1.95	1.45
Milc	1.73	1.45
Leslie3d	1.58	1.30
Bwaves	1.49	1.30
Soplex	1.34	1.30
Sphinx3	1.03	1.30
Wrf	0.95	1.19
CactusADM	0.68	1.19
Deall	0.62	1.19
Zeusmp	0.56	1.19
Tonto	0.55	1.19
Calculix	0.12	1.19
Namd	0.04	1.19
Gamess	0.00	1.19
Gromacs	0.00	1.19
Povray	0.00	1.19



Summary/Conclusions

- Balance is complicated and often counterintuitive
- It pays to develop a clear understanding of your own estimates of the “value” of a computation as a function of the time-to-solution and other factors
- These examples show that the “optimum” balance of a hardware system does not need to match the “application balance” in order to give the most cost-effective performance
 - Equal partitioning of execution time and equal partitioning of system cost are both sub-optimal solutions
 - The largest component of the Cost*Time equation is often the depreciation of the Base Cost during the execution of the application
 - This argues against using only CPU Cost and Memory Bandwidth Cost when optimizing system configurations





System Performance "Balance",
System Cost "Balance",
Application "Balance"
& the SPEC CPU2000/CPU2006 Benchmarks

John D. McCalpin, Ph.D.
Principal Scientist, AMD

Presented by Joerg Borowski, AMD

The author would like to thank the audience in advance for not lynching the presenter, who agreed to make this presentation without knowledge of the author's long and "interesting" relationship with the SPEC CPU committee. 😊

Today's Talk

- "System Balance", "Application Balance", and "Optimization" are commonly used terms, but are typically not clearly defined
 - Definitions
 - Examples
 - Performance Models
- "Price/Performance" is a popular metric for system optimization
 - Define cost (price) model
 - Combine with previous performance models
 - Discuss non-intuitive results
- Application to the SPEC CPU2000 FP and SPEC CPU2006 FP benchmarks
 - Theoretical price/performance optimization
 - "Real World" price/performance optimization

2

I am realizing that this topic has a lot more background than I initially anticipated. ☹

What are “System Balance” and “Application Balance”?



- Webster’s Online Dictionary has 9 definitions of “balance” as a noun. The two interesting ones are:
 - **5 a** : stability produced by even distribution of weight on each side of the vertical axis **b** : equipoise between contrasting, opposing, or interacting elements [*quantitative, but limited to one dimension*]
 - **6 a** : an aesthetically pleasing integration of elements [*not quantitative*]
- So when we talk about computer “balance” or application “balance”, do we know what we are talking about? [*Usually not*]
- Creating your own definitions of “balance” can be educational
 - How many variables? Which ones? How are they weighted?
 - Performance: CPU, cache, DRAM capacity, DRAM bandwidth, I/O?
 - Physical Attributes: Size, Power, Expandability, Serviceability?
 - Cost: Purchase, Support, Power/Cooling?
- Today’s talk focuses on a typical definition for “System Balance” and “Application Balance”
 - Balance = (Bytes/second of memory BW) / (CPU Cycles of computation)



This is drawn from the introduction of the paper I recently published in the CTWatch Quarterly web journal:

<http://www.ctwatch.org/quarterly/archives/february-2007/>

The article is “The Role of Multicore Processors in the Evolution of General-Purpose Computing” by **John McCalpin, Chuck Moore, Phil Hester**, (all Advanced Micro Devices, Inc.)

A PDF version of the issue is downloadable:

<http://www.ctwatch.org/quarterly/pdf/download.php?file=ctwatchquarterly-10.pdf>

Although most people understand that they are using the word “balance” in an imprecise manner, unless you sit down and try to work out real quantitative definitions I don’t think that you are likely to have a very good idea of just *how* imprecise this word is!

Just looking at the few attributes on this page suggests a frighteningly large space of possible reasonable definitions (as well as an even larger space of unreasonable definitions).

Even the “quantitative” definition on the bottom is very fuzzy. Are we talking about “peak memory bandwidth”, “maximum sustainable memory bandwidth”, “bandwidth actually sustained while running some specific code” or something else? Ditto for “FLOP/s of compute capability”: Peak? Max sustainable for an arbitrary combination of operations? Max sustainable for the specific combination of operations in my application? Actual delivered FLOP/s on my application?

What is “Optimization”

- Another very tricky word...
- Webster’s online:
 - : an act, process, or methodology of making something (as a design, system, or decision) as fully perfect, functional, or effective as possible; *specifically* : the mathematical procedures (as finding the maximum of a function) involved in this
- “Optimization” does not mean making something “better” in the abstract – it means maximizing or minimizing a specific objective function
- What is **your** objective function?
 - Optimize [single-thread, multi-thread, throughput] x [performance, performance/cost, performance/watt]
 - Satisfy constraints: [max hardware cost, max physical size, minimum memory, minimum I/O, expandability, max allowable time-to-solution, software compatibility]
 - I count at least 20 primary purchasing metrics in common use for servers



Also from CTWatch paper. Again we have a disturbingly large space of potential definitions.

“Optimization” (continued)

- Computing is a human activity that can be modeled as a rational economic decision
- The fundamental metrics are therefore
 - **Value** – what *benefits* do I obtain from doing this computing?
 - **Cost** – what do I have to *give up* to do this computing?
- These may be only loosely related to technology metrics like “performance” or “price”
 - E.g., “Value” is typically bounded as time-to-solution goes to zero – i.e., as “performance” goes to infinity
 - E.g., “Value” is typically bounded as “price” goes to zero – how many computers can I use, even if they are “free”?
- I can’t define your “Value” or “Cost” metrics
 - So my examples will be based on “Performance” and “Price”
 - This does not negate the critical importance of the differences between “Value” and “Performance” & “Cost” and “Price” !



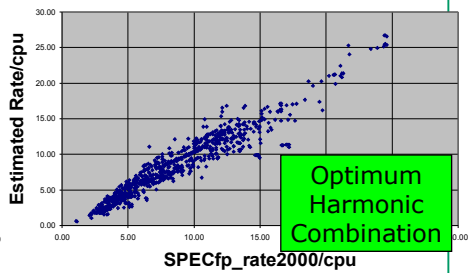
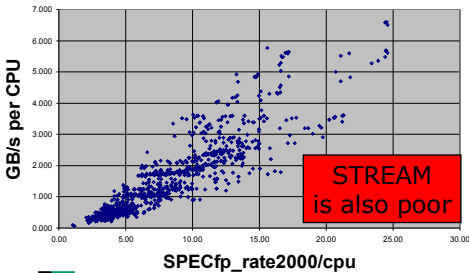
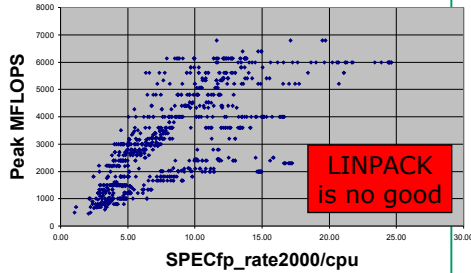
These notes are based on a 1997 presentation that I gave at a NASA workshop on “Performance Engineered Systems for the Information Power Grid”. The main point is that performance improvement by parallelization is only valuable when the improved “value” of quicker time to solution is “bigger” than the increased “cost” due to the use of parallel systems (that have larger “price” and less effective use of resources due to imperfect parallel performance scaling).

Note that this argument ignores one of the most important motivations for parallel computing – the ability to run simulations that are too large to fit into the memory of smaller systems. In this case the time-to-solution is effectively infinite (or the performance is zero) for smaller systems. In this case the baseline for analysis of “Value” and “Cost” for consideration of using larger systems should be the smallest parallel system that can actually run the application of interest.

A Simple Performance Model



Consider the task of projecting SPECfp_rate2000 using as few parameters and/or measurements as possible
"McCalpin's Magical Metric" obtains ~16% RMS error using 1 measurement plus 2 architectural parameters



Presentations available at <http://www.cs.virginia.edu/~mccalpin/>

The optimum harmonic combination that I like to refer to as "McCalpin's Magical Metric" was presented at SuperComputing2003 and SuperComputing2004.

Slides are available online at <http://www.cs.virginia.edu/~mccalpin/>

The model attempts to project SPECfp_rate2000/cpu using a simple model:

$$\text{Define a "pseudo-time": } T = 1/(\text{SPECfp_rate2000/cpu})$$

Assume two additive performance components

$$T = W_{\text{cpu}}/R_{\text{cpu}} + W_{\text{bw}}/R_{\text{bw}}$$

R_{cpu} is assumed to be equal to the peak GFLOPS

R_{bw} is assumed to be equal to the sustainable memory bandwidth per SPEC "user"

W_{bw} is a 3-coefficient model based on cache size, with coefficients fit to the data by a least squares error minimization process. (See the original presentations for details.)

Slightly Less Simple Performance Models

- “McCalpin’s Magical Metric” can be made slightly more accurate by developing separate models for each of the 14 benchmarks and then taking the geometric mean of the results
 - It is very hard to get to 10% accuracy without adding many more degrees of freedom to the model or adding significantly more parameters to describe microarchitectures and/or software
- Dramatically more accurate models can be made for systems based on a single processor family and fixed compiler choice
- Here I consider models for SPEC CFP2000 and CFP2006 (speed and rate) for systems using AMD Opteron processors
 - All systems have the same CPU microarchitecture and same cache sizes
 - All systems are running “similar” versions of Linux and effectively identical compilers (PathScale EKOPath 2.5 compiler suite)
 - Each of the 14 SPECfp2000 and 17 SPECfp2006 codes has its own model
- Two parameter models are not flexible enough to get <10% errors
 - Adding a “Memory Latency” component reduces errors to <<5%



Many years of work on SGI MIPS, SGI Itanium, IBM POWER, and AMD Opteron systems is skipped over here in the interests of brevity. Each system has slightly different sharing characteristics (shared memory interfaces, shared front-side busses, shared caches, shared cores (e.g., simultaneous multithreading)), requiring the development of unique models for each major variation.

While many models can provide accurate interpolation, models based on weighted harmonic means have correct asymptotic properties and actually become more accurate as the system configurations become more “unbalanced”. Confidence in a model’s ability to extrapolate is critical for support of design and business decisions.

The models are also very useful for “sanity checking” of benchmark measurements. While at IBM, I developed the models to the point where they were significantly more accurate than a single run of the actual benchmarks. Deviations of >2% on any of the individual benchmarks almost always indicated a configuration error with the benchmark rather than a projection error of the model.

Slightly Less Simple Model, continued



- The model is additive (ignores overlap) and is of the form:

$$T = \frac{W_{CPU}}{R_{CPU}} + \frac{W_{BW}}{R_{BW}} + \frac{W_{latency}}{R_{latency}}$$

- The three "Rate" parameters are specified *a priori* based on the system configuration:

Metric	Description	Value on Reference System ⁽¹⁾
R_{CPU}	CPU GHz	2.8 GHz
R_{BW}	Sustained GB/s per SPEC "user"	2.6 GB/s per "user"
$R_{latency}$	Idle pointer-chasing memory latency of system	60 ns

⁽¹⁾ Reference Hardware Configuration: Two socket server with two 2.8 GHz dual-core AMD 2000 series processors & four 667 MHz DDR2 DIMMs per socket



The Rate components are prescribed based on each hardware configuration. The (total) Time is based on published measurements for each benchmark on each configuration (see next slide).

The unknown variables are the three Work components for each of the 31 benchmarks (14 CFP2000 plus 17 CFP2006).

Note that since all Opteron caches are the same size and no caches are shared, the Work components can be assumed to be independent of the hardware configuration (single vs dual core, etc).

The methodology becomes much more tricky when the Work components are different for each hardware configuration due to cache sharing, simultaneous multithreading, etc.

The "Work" components might not mean exactly what they appear to mean. W_{cpu} means the amount of work done at a rate that is proportional to the CPU frequency. This includes the CPU core, the L1 and L2 caches, and portions of the Opteron memory controller. Similarly W_{bw} means the amount of work done at a rate that is proportional to the sustained memory bandwidth per benchmark process (based on the performance of the STREAM benchmark and the 171.swim benchmark from SPEC CFP2000). W_{lat} means the amount of work that is proportional to the idle memory latency of the system (60 ns on 1 or 2 sockets systems and 90 ns on 4 socket systems).

Slightly Less Simple Model, continued again



- The three “Work” components are based on a least-squares fit of the published execution time to the model projection for each benchmark
 - Separate models for each benchmark (14 CFP2000 + 17 CFP2006)
- Published Data
 - CPU2000: 9 “speed”, 20 “rate”, 6 independent system configurations
 - CPU2006: 7 “speed”, 22 “rate”, 13 independent system configurations
- The models are strongly consistent with the measured results:

RMS error of projected execution time vs published results	CFP2006 17 benchmarks	CFP2000 14 benchmarks
4% - 6%	2 (12%)	0 (0%)
2% - 4%	7 (41%)	4 (29%)
0% - 2%	8 (47%)	10 (71%)

9

“Independent System Configurations” means unique combinations of frequency, bandwidth per SPEC “user”, number of sockets (2 and 4 sockets have different idle memory latency: 60 & 90 ns), etc.

The graphs at the end of the presentation show strong correlation between the model projections and the measured times, with only a few cases that show suggestions of systematic error (e.g., bwaves and lbm).

For CPU2000, I only had 6 independent configurations for the 3 degrees of freedom, so these results are probably somewhat sensitive to experimental “noise” in the performance measurements.

For CPU2006, the 13 independent system configurations provide strong constraints for the 3 degrees of freedom, but even in this case a few of the configurations were represented by only one or two submissions. A number of clearly sub-optimal results were removed before I fit the models to the data, but some unexplained variability remains.

When I do this work for “real”, I prefer to perform all the measurements myself to control for system configuration and other sources of variability in the performance measurements, but these results are clearly good enough to show the broad range of “application balance” across the SPEC FP2000/FP2006 benchmarks.

The effectiveness of the model at fitting the measurements is a strong indicator that the basic model assumptions are correct: (1) CPU work, Memory BW work, and Memory Latency work are largely additive, rather than overlapped; (2) Memory time can be effectively modeled as the sum of prefetchable component (limited by the max sustainable BW of the chip) and a non-prefetchable component (limited by the idle memory latency of the system).

It is possible that a different set of primary Work and Rate components would be even more effective, but I have worked extensively with 2-term models and I do not believe that they are capable of this level of accuracy. These projection errors are so small relative to the “natural” variability of the measurements that it would be difficult to demonstrate that a different 3-term model (or a more complex model) is “better” than this one.

CFP2006 Execution Time on Reference System



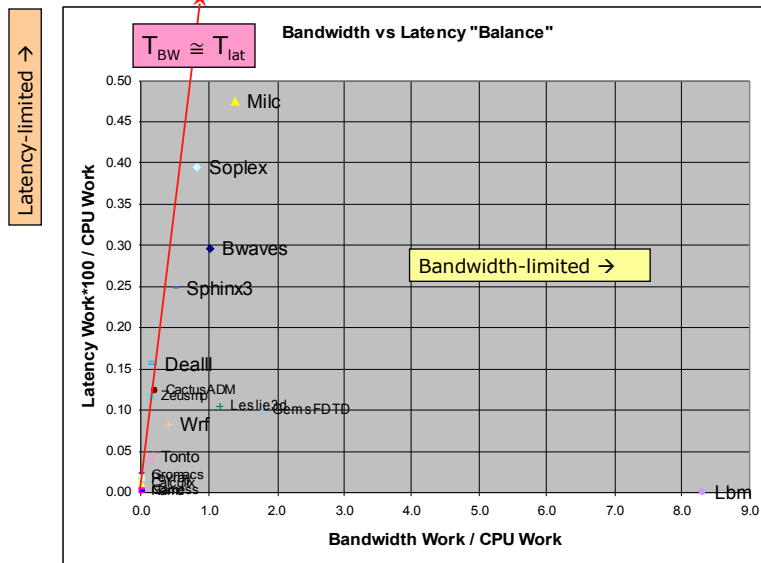
Again, the Reference System is a 2 socket Opteron RevF box with two dual-core 2.8 GHz Opteron processors and a full load of DDR2/667 DIMMs, running 4 copies of each benchmark in the SPEC_rate configuration.

(CAS latency does not appear to impact performance on any of these benchmarks, but I believe that most of the measurements were done with CAS 5 DIMMs.)

Although the Latency times are small in many of these codes, recall that this is a machine with 60 ns local latency – most machines have significantly higher idle memory latency and should show correspondingly higher memory latency stall times.

The largest errors in the performance projections were for the benchmarks with the highest memory times. From looking at the data it appears that this represents both systematic and random components. The run-to-run and submission-to-submission variability is definitely largest for the high-bandwidth benchmarks. Also the assumption that “bandwidth” is a single entity is a crude one. Sustained bandwidth varies significantly based on prefetchability of the address streams, the ratio of reads to writes, etc. The strong (though not perfect) correlation between the model projections and the observations suggests that within the high-bandwidth subset of these codes, treating “bandwidth” as a single parameter is not a bad approximation. It may be a terrible approximation for the low-bandwidth codes, but since memory bandwidth time contributes such a small portion to their execution time, any such systematic errors would be irrelevant to the accuracy of the projections.

Latency and Bandwidth "Balance" vs CPU work



11

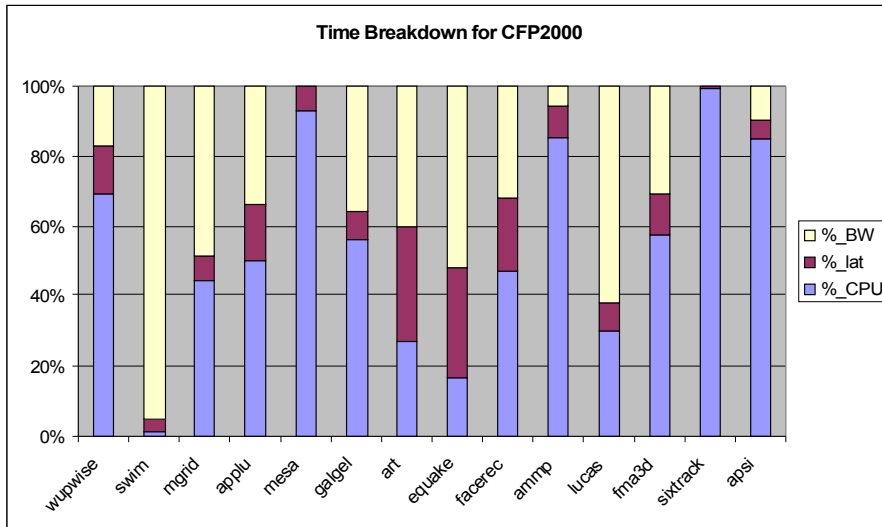
For the reference system, one latency of 60 ns corresponds to a transfer of about 156 Bytes at the nominal "streaming" bandwidth of 2.6 GB/s per "user".

For the Reference System, benchmarks to the right of the red line spend more time waiting on bandwidth-limited memory accesses than on latency-limited memory accesses.

- About 5 of 17 codes have non-trivial memory latency stall times that are similar to the memory bandwidth stall times. 5 of the 17 have memory bandwidth stall times that significantly exceed latency stall times. The remaining seven codes are almost completely CPU-time dominated.
- Only 5 of 17 codes appear to require more than "1 Byte/Cycle" of Memory Bandwidth and only 1 code requires >2 Bytes/Cycle

Of course these breakdowns only apply to systems with "system balance" similar to the reference system, caches similar to the 1 MB/"user" of the reference system, and compilers with high-level optimizations similar to those of the PathScale EKOPath 2.5 compiler suite.

CFP2000 Execution Time on Reference System



12

Same as before but for the SPECfp2000 benchmarks. Same reference system: 4 user SPECfp_rate2000 on 2-socket, dual-core, 2.8 GHz Opteron 2x00 with DDR2/667 memory.

The ratios for the 171.swim benchmark are not entirely reliable, as I “tweaked” the bandwidth “Rate” values for the various configurations to better project the SWIM benchmark results. This amounts to assuming that SWIM is almost 100% memory time – not a bad assumption, but not a perfect one either. The published SPECfp_rate2000 results don’t provide an adequate variety of CPU frequencies to obtain a clear differentiation between CPU time and Memory BW time here.

Both sets of benchmarks display the full range of CPU-limited to Bandwidth-limited codes, along with some codes that have significant latency stall components. Overall the CPU2006 suite has a few more CPU-intensive codes than the CPU2000 suite, moving it closer to the distribution of “important” “real” applications that I reported in my WWC2000 keynote presentation:

<http://www.cs.virginia.edu/~mccalpin/wwc-keynote.html>

Performance Model Summary

- Simple performance models with a few additive components can be fit very accurately to each of the SPEC CFP2000_rate and SPEC CFP2006_rate benchmarks
- The SPEC CFP2000 and CFP2006 benchmarks both show a very broad range of "Application Balance", ranging from CPU-limited to (almost) Memory Bandwidth-limited, with a substantial subset showing non-trivial Memory Latency stall times.
- The broad range of these results and the accuracy of the models suggest that the models will be quite useful in further modeling of Price/Performance

13

OK, everybody take a deep breath....

Now on to the Price side of the equation.

Cost Model

- In principal, additive cost models can be constructed for computer systems
- Difficulties & Limitations
 - Costs can vary dramatically due to features that do not impact performance (e.g., reliability, scalability, low power, etc)
 - Many theoretically possible configurations are not available because no one has chosen to build them
- Key Feature: Cost models have a large “baseline” component associated with the “stuff” necessary to support the CPU and memory (motherboards, disk, power supplies, etc)
- For the initial analytical modeling, I will assume:

$$C = C_{base} + \$_{CPU} R_{CPU} + \$_{BW} R_{BW}$$

- C = cost, R = rate
- $\$_{CPU}$ = cost per unit CPU performance
- $\$_{BW}$ = cost per unit sustainable memory BW

14

I am using “Cost” and “Price” interchangeably here. The differences are important, but belong in a different talk.

The assumption that CPU and BW “Cost” and “Performance” are linearly related is of limited applicability. System configuration limitations, minimum CPU frequencies, and minimum memory bandwidths (associated with the 2 GB/”user” DRAM capacity requirement for CFP2006) act to reduce the customer’s ability to customize system configurations to specific targets. The combination of these limitations is confusing, but ultimately not really a problem – the linear model here is enough to show *why* the optimum system balance is not what you might expect – detailed system optimization should be based on real-world configurations with real-world pricing, as presented later in this talk.

On the Opteron systems that I am using for this modeling, there is no way to pay more money to reduce the memory latency (by any significant amount), so latency is ignored in the cost model and is effectively a constant in the performance model (i.e., only R_{bw} and R_{cpu} are free variables in the system configuration).

Cost/Performance

- Cost/Performance \approx Cost * Time

$$C = C_{base} + \$_{CPU} R_{CPU} + \$_{BW} R_{BW} \quad \times \quad T = \frac{W_{CPU}}{R_{CPU}} + \frac{W_{BW}}{R_{BW}} + \frac{W_{latency}}{R_{latency}}$$

- or

$$C = C_{base} + C_{CPU} + C_{BW} \quad \times \quad T = T_{CPU} + T_{BW} + T_{latency}$$

- Note that Cost * Time has 9 terms!
 - Two terms correspond to "active" useful work: $T_{cpu} * C_{cpu}$ and $T_{bw} * C_{bw}$
 - The remaining 7 terms are overheads:
 - Depreciation of active components while other active components are busy
 $T_{cpu} * C_{bw}$ $T_{bw} * C_{cpu}$
 - Depreciation of inactive components while the active components are busy
 $T_{cpu} * C_{base}$ $T_{bw} * C_{base}$
 - Depreciation of everything while waiting on memory latency stalls
 $T_{latency} * C_{base}$ $T_{latency} * C_{cpu}$ $T_{latency} * C_{bw}$

15

The first two terms of the Cost*Time product correspond to the cost of depreciation of the "active" components of the system while they are busy doing work.

It is the existence of these seven overhead terms that make Cost/Performance optimization counterintuitive.

Notice that I have added "Depreciation" as another synonym for "Cost" and "Price". This term helps remind us that systems with finite "life spans" are depreciating every second whether they are being used or not. Faster CPUs and more Bandwidth reduce run-time and so decrease the (large) depreciation of the Base cost during the execution of the job. On the other hand, too much of either CPU or Bandwidth results in excessive depreciation of these expensive components while the other underprovisioned component takes a long time to finish its part of the work.

Optimization of Cost/Performance

- Define some nondimensional parameters:

$$\beta = R_{bw}/R_{cpu} \quad \leftarrow \text{machine balance}$$

$$\gamma = W_{bw}/W_{cpu} \quad \leftarrow \text{application balance}$$

$$\delta = \$_{bw}/\$_{cpu} \quad \leftarrow \text{technology cost balance}$$

- In the absence of the constant terms (C_{base} & $T_{latency}$), Cost*Time is a simple quadratic function with a simple analytic optimum

$$\beta_{optimum} = \sqrt{\frac{\gamma}{\delta}}$$

- In the presence of the constant terms the equation is effectively quadratic and the one physically realizable optimum provides specific values of R_{cpu} and R_{bw}

$$R_{CPU} = \sqrt{\frac{W_{CPU}}{\$_{CPU}}} \times \sqrt{\frac{C_{base}}{T_{latency}}} \quad R_{BW} = \sqrt{\frac{W_{BW}}{\$_{BW}}} \times \sqrt{\frac{C_{base}}{T_{latency}}}$$

Interestingly, the optimum machine balance ratio β is the same in both cases

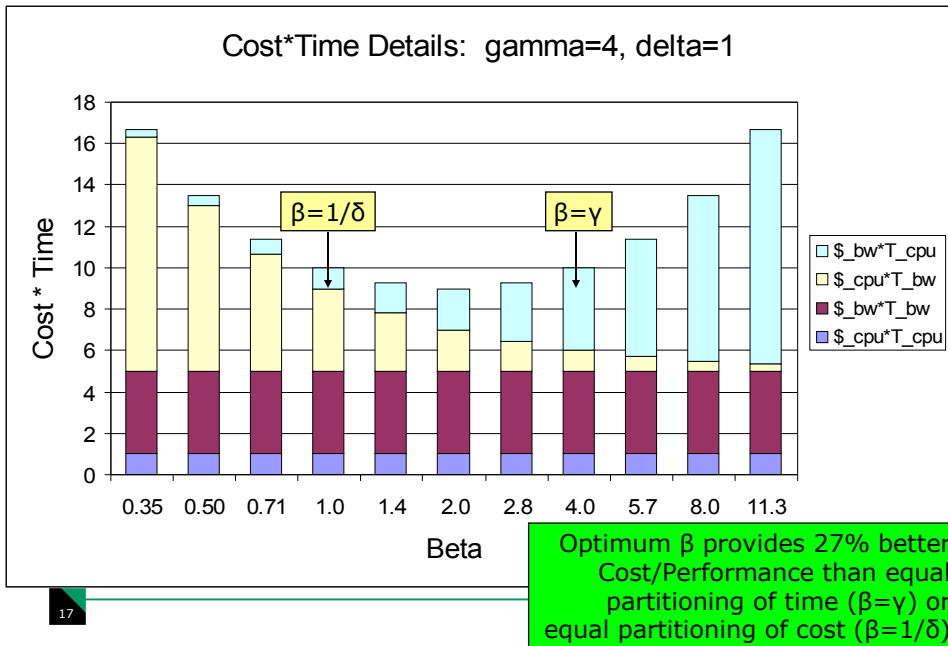
16

The algebra is left as an exercise for the reader.

To minimize Cost*Time you must differentiate with respect to both R_{bw} and R_{cpu} , set both to zero and combine into a single equation in terms of either of these variables. In the presence of the constant terms this equation is actually a fifth-order polynomial, but one of the roots is [$R_{bw}=0$, $R_{cpu}=0$] which clearly corresponds to a *maximum* of Cost*Time (due to infinite Time), and can be factored out leaving a fourth-order polynomial.

Fortunately this fourth-order polynomial has nice symmetries, with the four solutions given by the four +/-, -/+ combinations of the values above. Only the +, + root makes physical sense (although negative computation rates could lead to some interesting theoretical physics).

A Simple Analytical Example (no constant terms)



This example considers the case of a high-bandwidth application (4 Bytes/Cycle) and assumes that Compute and Bandwidth have similar costs per unit performance ($\delta=1$).

Without the constant terms there are only 4 terms in the Cost*Time product.

Recall: γ = “Application Balance” (W_{bw}/W_{cpu}), β = “System Balance” (R_{bw}/R_{cpu}), δ = “Technology Cost Balance” ($\$/\$$).

In this case the analytical solution is $\beta = \sqrt{\gamma/\delta} = \sqrt{4/1} = 2$

The “traditional” optimization approach of setting $\beta=\gamma$ is suboptimal because the job is spending $1/2$ its time on Compute and $1/2$ on Bandwidth, but only $1/4$ of the system cost was for Compute. Therefore you can improve the Cost*Time by purchasing more Compute to decrease that 50% of the execution time.

An alternate “optimization” might be setting $\beta = 1/\delta$. This makes the Cost of Compute and the Cost of Bandwidth equal. This is suboptimal because you have spent 50% of your money on Compute, but only 25% of your execution time is spent on Compute – you should buy less.

These two traditional optimizations are equally sub-optimal, corresponding to symmetric offsets from the optimum on the parabolic curve of Cost*Time.

Think about how much CPU frequency or Memory Bandwidth you would have to add to a system (at fixed price) to get a 27% Performance improvement. These are not trivial differences!

More detailed examples

- Given that we do have computers, we do not need to limit ourselves to analytical solutions.
- I obtained pricing for a wide variety of 1U, 2-socket Opteron servers from www.serversdirect.com during June 2007
 - Opteron “RevE” single-core CPUs @ 5 frequencies, with DDR/400 memory
 - Opteron “RevE” dual-core CPUs @ 6 frequencies, with DDR/400 memory
 - Opteron “RevF” dual-core CPUs @ 6 frequencies, with DDR2/667 memory
- The pricing for the servers was split into “base”, “CPU”, and “Memory” components
 - Yes, I am deliberately confusing “Memory Cost” and “Bandwidth Cost”, but I don’t have time to explain why this is less wrong than it might appear
 - Each system was configured with the cheapest option providing 2 GB/core
- Cost*Time was computed for each of the 31 benchmarks on each of the systems, and the configuration with the lowest Cost*Time was identified



All the words are included on the slide....

Optimum Price/Performance for Opteron "RevE"



CPU/Memory	Cores/Chip	CPU GHz	Optimum for this many of the 14 CFP2000 benchmarks	Optimum for this many of the 17 CFP2006 benchmarks
Opteron RevE, DDR/400	1	2.2	1	-
Opteron RevE, DDR/400	1	2.4	-	1
Opteron RevE, DDR/400	2	1.8	2	2
Opteron RevE, DDR/400	2	2.0	2	3
Opteron RevE, DDR/400	2	2.2	9	11

19

I am presented the results for RevE and RevF separately because the newer RevF provides better Price/Performance on all the benchmarks, but only the RevE provides the single-core option for Price comparison.

Of the 11 RevE configurations, only these five produced a minimum Cost*Time for any of the 31 benchmarks when using the web site's price and my detailed 3-term performance model.

The single-core chips provided optimum Price/Performance for 171.swim from SPECfp2000 and 470.lbm from SPECfp2006.

Optimum Price/Performance for Opteron "RevF"



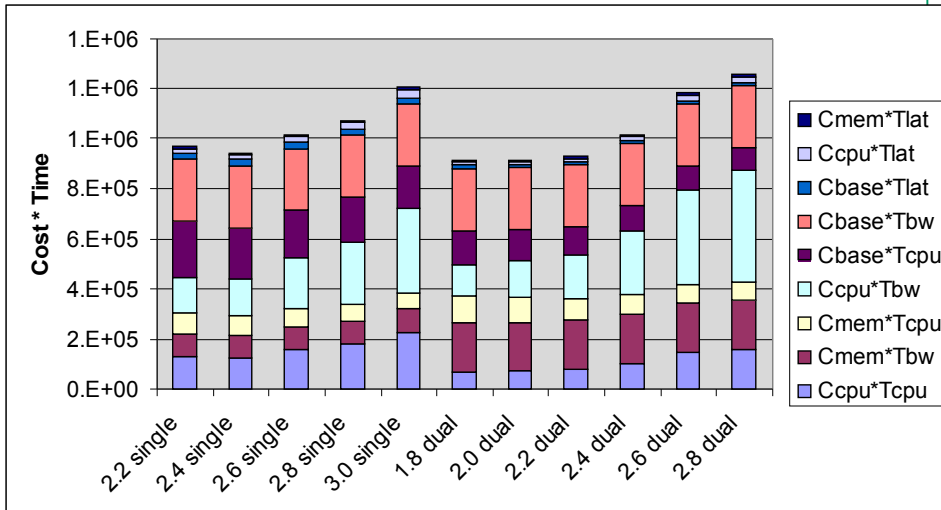
CPU/Memory	Cores/Chip	CPU GHz	Optimum for this many of the 14 CFP2000 benchmarks	Optimum for this many of the 17 CFP2006 benchmarks
Opteron RevF, DDR2/667	2	1.8	3	3
Opteron RevF, DDR2/667	2	2.0	3	4
Opteron RevF, DDR2/667	2	2.2	8	10

- Note that the Performance model can provide very accurate performance projections for both single-core and dual-core parts, but...
- AMD is not currently offering single-core RevF Opteron processors, so pricing is only available for the dual-core processors here



Of the 6 RevE configurations, only these 3 produced a minimum Cost*Time for any of the 31 benchmarks when using the web site's price and my detailed 3-term performance model.

Detailed Cost*Time Breakdown for GemsFDTD



21

Actual Cost * Modeled Time for RevE processors.

GemsFDTD is the code with the second-highest bandwidth requirement (almost 2 bytes/Cycle), so it spends a large fraction of its time doing memory work. The largest components of the depreciation are therefore those associated with Tbw.

It clearly does not pay to put in expensive CPUs (e.g., the last 2 columns on the right), as this results in large Ccpu*Tbw, i.e., CPU depreciation during memory bandwidth time.

This code might represent an opportunity for faster/more expensive DRAMs, if they were supported by the system. This code is sufficiently bandwidth-limited that single-core and dual-core systems have very similar Cost*Time, with the dual-core systems taking 1.55 to 1.59 times as long to complete 4 jobs as the single-core systems take to complete 2 jobs.

Failure and Success with Analytical Modeling



- Linear trends were fit to the CPU and memory cost data to create an analytically tractable cost model
- Optimum R_{cpu} and R_{bw} were computed for each of the 17 CFP2000 benchmarks
 - For 8 benchmarks the computed optimal R_{cpu} and R_{bw} were within physically realizable limits for systems based on the Opteron processor
 - For 3 benchmarks the computed optimal R_{bw} was much higher than is currently configurable
 - These benchmarks might benefit from single-core processors
 - For 6 benchmarks the computed optimal R_{cpu} was much higher than is currently configurable
 - These benchmarks should benefit from quad-core processors
- Bad News: poor numerical correlation between analytical and observed optimum system balance
- Good News: perfect correlation of rank ordering of analytical and observed optimum system balance

22

As noted previously, the assumption of a linear dependence between cost and performance is not particularly useful. Processor prices rise much faster than linearly with frequency and memory bandwidth is maximized in all configurations with enough DRAM to hold the benchmarks – there is no way to spend less money and get less bandwidth for this particular set of system configurations (though the correlation must exist when considering a broader design space).

Higher-order cost models are not analytically tractable because optimization of linear cost * linear time already generates a quadratic equation. One would have to be either very clever or very lucky to find a nonlinear cost model that resulted in the generation of a factorable or otherwise reducible optimization problem.

Benchmark	Analytic Optimum	Observed Optimum
	Beta	Beta
Lbm	4.03	1.45
GemsFDTD	1.95	1.45
Milc	1.73	1.45
Leslie3d	1.58	1.30
Bwaves	1.49	1.30
Soplex	1.34	1.30
Sphinx3	1.03	1.30
Wrf	0.95	1.19
CactusADM	0.68	1.19
Dealll	0.62	1.19
Zeusmp	0.56	1.19
Tonto	0.55	1.19
Calculix	0.12	1.19
Namd	0.04	1.19
Gamess	0.00	1.19
Gromacs	0.00	1.19
Povray	0.00	1.19

23

Comparison of the Analytical optimum system balance based on a linear approximation to the cost model and the observed optimum system balance based on the actual price * modeled performance.

The perfect correlation of rank-ordering is encouraging, as is the good correlation between the numerical values at which the observed optimum beta switches from one value to another.

The linear approximation to the cost model is far too coarse to justify more detailed comparisons – presumably values could be chosen to make this comparison look even better, but the first try looks good enough to support the concept of cost/performance optimization using the product of a cost model and a performance model.

Summary/Conclusions

- Balance is complicated and often counterintuitive
- It pays to develop a clear understanding of your own estimates of the “value” of a computation as a function of the time-to-solution and other factors
- These examples show that the “optimum” balance of a hardware system does not need to match the “application balance” in order to give the most cost-effective performance
 - Equal partitioning of execution time and equal partitioning of system cost are both sub-optimal solutions
 - The largest component of the Cost*Time equation is often the depreciation of the Base Cost during the execution of the application
 - This argues against using only CPU Cost and Memory Bandwidth Cost when optimizing system configurations

24

Wake up now, the presentation is finished....