IBM Research

A Holistic Approach towards Automatic Performance Analysis and Tuning

Advanced Computing Technology
IBM T.J. Watson Research Center
ihchung@us.ibm.com

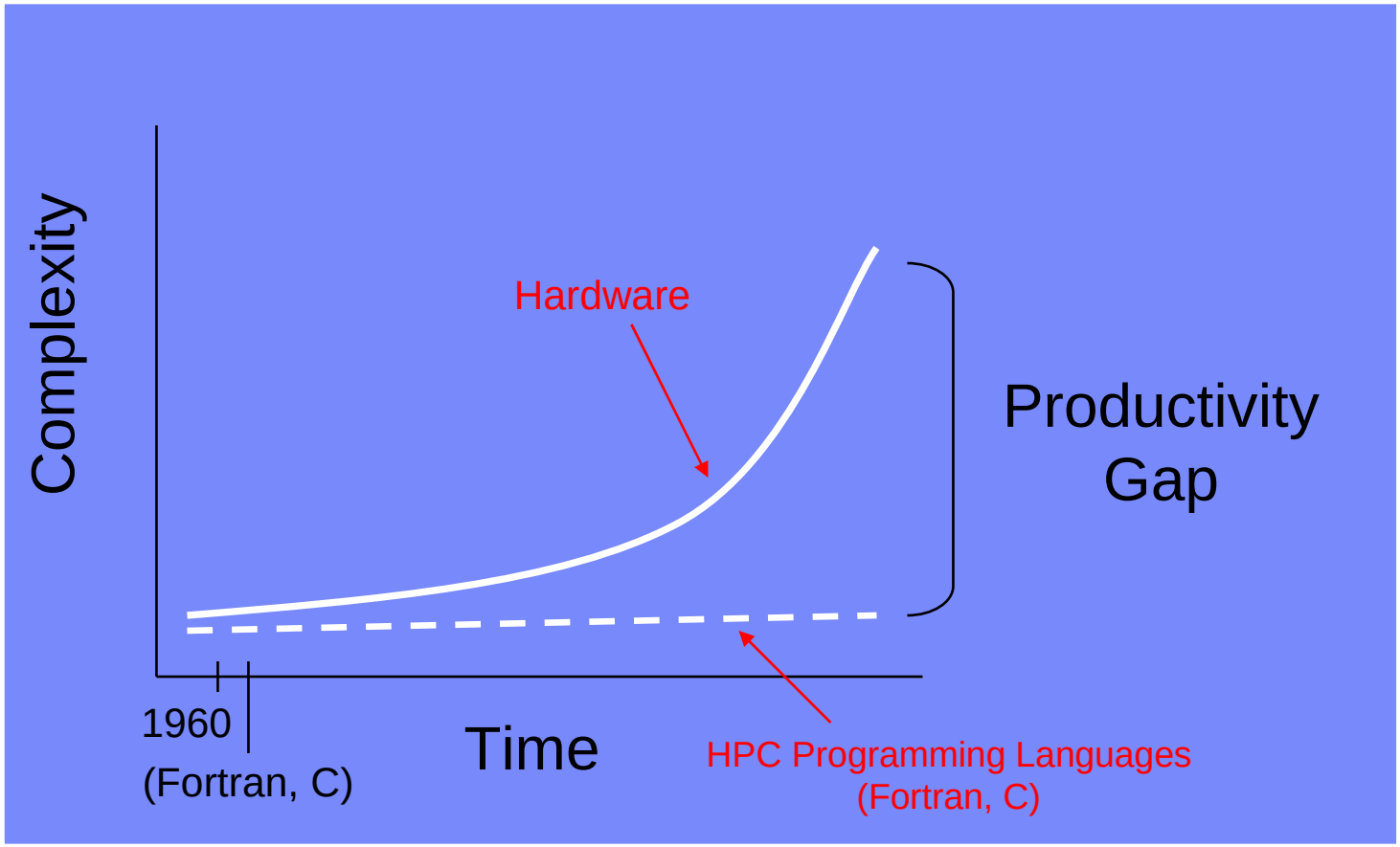
© 2009 IBM Corporation

System Evolution

- Device Scaling imposing fundamental constraints on system
  - Power dissipation and energy consumption
  - Physical size / packaging
- Pressure to re-think system architecture
  - Blue Gene: low power devices, embedded (small)
  - Cell: Attached (embedded) co-processing engine
- Systems become inherently more complex
  - Connectivity / hierarchical topology (torus, intra-cell)
  - Multi-core processors (and less memory per processor)
  - Multi-thread (SMT, hyperthreading)
- This poses new challenge to application programming
  - New programming paradigm? (but ~$1T in legacy codes, ISV apps, etc.)
- Conclusion: New software tools essential to mitigate evolving system complexity and improve productivity.

© 2009 IBM Corporation

Enablement Productivity Gap = Hardware – Software



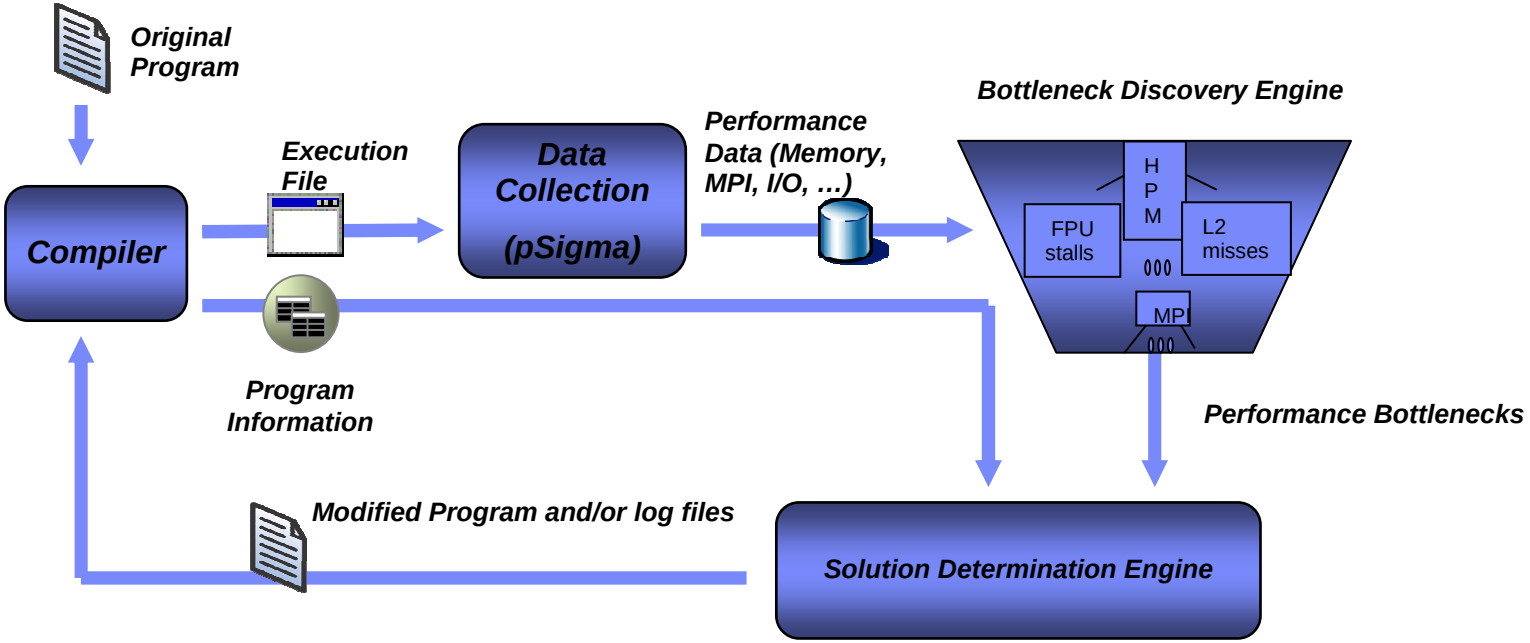
© 2009 IBM Corporation

PERCS Impact on Productivity Gap

- State-of-Art Application Enablement circa 2002+
  - Source code modification (e.g., timing routines)
  - Non-selective, non-source code correlated tools (e.g., PAPI)
  - Dynamic instrumentation via external agents (e.g., DynInst)
  - GUI frameworks to look at data (e.g., Vampir, Vtune, Tau)
  - No unified analysis framework (CPU, MPI, OpenMP, and I/O)
  - No management of large scale performance data
- IBM DARPA HPCS Toolkit
  - Next generation unified framework for automated (not automatic) intelligent-assist of application performance tuning including…
    - No source code modifications…but with source code correlation of the data
    - Selective and dynamic instrumentation without external agents
    - Large scale data management
- In a Nutshell:
  - Previous tools only show you the data…does not resolve the Productivity Gap.
  - The HPCS Toolkit makes sense of the data…closes the Productivity Gap.

© 2009 IBM Corporation

High Level Design Flow for HPCS Toolkit



- HPCS Toolkit provides Automated Framework for Performance Analysis.
  - Intelligent automation of performance evaluation and decision system.
  - Interactive capability with graphical/visual interface always available.

Bottleneck: elapsed time exceeds threshold for completing work.

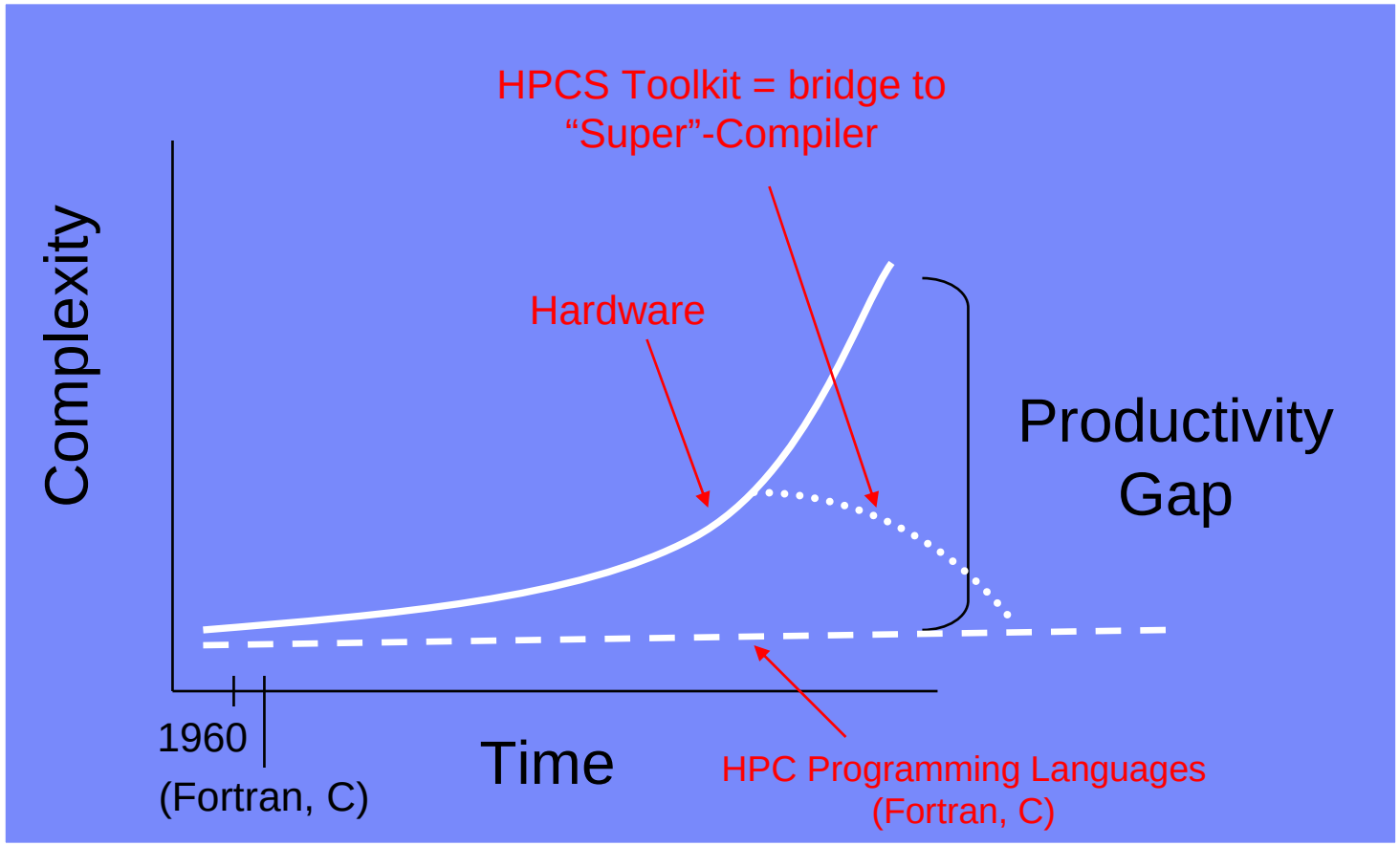
© 2009 IBM Corporation

HPCS Toolkit Scalability

- Self-Contained Performance Data Collection Framework
  - Part of the instrumented application executable
    - No background processes or external agents
    - Extensible to MRNet (University of Wisconsin)
- Use of Parallel File System (GPFS)
  - Data managed in parallel via distributed files
    - Up to five files per process (e.g., for each MPI task):
      1. HPM data
      2. MPI data
      3. OpenMP data
      4. Memory reference data
      5. I/O data
- Pre-runtime and Post-runtime Filtering Capability
  - User-defined logic to reduce data to be captured and/or analyzed
- IBM Research Blue Gene test-bed
  - Up to 0.5 million processor systems

© 2009 IBM Corporation

Closing the Enablement Productivity Gap



© 2009 IBM Corporation

Automated Performance Tuning – Timetable

2007 Deliverables:

- Performance Data Collection
  - Scalable, dynamic, programmable
  - Completely binary: no source code modification to instrument application…
  - But retains ability to correlate all performance data with source code
- Bottleneck Discovery
  - Make sense of the performance data
  - Mines the performance data to extract bottlenecks

FUTURE MILESTONE DELIVERABLES:

- Solution Determination - 2008 - 2009
  - Make sense of the bottlenecks
  - Mines bottlenecks and suggests system solutions (hardware and/or software)
  - Assist compiler optimization (including custom code transformations)
- Performance "Visualization" - 2008 - 2010
  - Performance Data / Bottleneck / Solution Information feedback to User
    - Logging (textual information)
    - Compiler feedback
  - Output to other tools (e.g., Kojak analysis, Paraver visualization, Tau, etc.)

© 2009 IBM Corporation

Typical Tuning Life Cycle

- Observing behavior, formulating hypothesis, conducting validation tests
  - Application instrumentation for performance data collection
  - Correlate performance data with the program characteristics
  - Trace back to the source program
- Optimization to improve performance

© 2009 IBM Corporation

Performance Diagnosis

- Requirement
  - In depth knowledge of Algorithm, Architecture, Compiler, Run time behavior
- Performance data
  - Collecting, Filtering, Searching, Interpreting
- Coordinating multiple components of a complex system
- Challenging and time consuming even for experienced users

© 2009 IBM Corporation

Performance Optimization Strategy

- A framework provides
  - Performance data collection
  - Bottleneck identification
  - Solution discovery
  - Implementation
  - Iteration of the tuning process
- Key components
  - Performance tools
  - Compiler
  - Expert knowledge

© 2009 IBM Corporation

Performance data

- Wide array of information
  - Static analysis
  - Runtime behavior
  - Algorithm property
  - Architecture feature
  - Expert knowledge
- Correlate performance metrics from different aspects
  - Computation
  - Memory
  - Communication
  - I/O

© 2009 IBM Corporation

Bottleneck Discovery

- Bottleneck is part of the system that limits the performance
- A mechanism to mining the expert knowledge is necessary to automate the tuning process
  - Wisdom is often expressed in fuzzy terms
- Example
  - MPI derived data type for data packing
  - Detect packing behavior
    - Identify the buffer being sent (MPI tracing)
    - Runtime memory access analysis (intercepting loads/stores)
    - Flow analysis (via static analysis)

© 2009 IBM Corporation

Bottleneck Discovery (continue)

- A bottleneck
  - A rule (pattern) defined on a set of metrics
  - Currently is a logic expression
  - Provides a way to compare and correlate metrics from multiple sources and dimensions
- A performance metric is any quantifiable aspect about or related to application performance. For example,
  - Number of pipeline stalls for a given loop
  - Number prefetchable streams
  - Number of packets sent from a certain processor
  - Size of physical memory

© 2009 IBM Corporation

Metrics from existing performance tools

Metric name	Description	Collected by
PM_INST_CMPL	Instruction completed	HPM
L1_miss_rate	L1 miss rate	HPM
Avg_msg_size	Average message size	MPI profiler
Thread_imbalance	Thread work load imbalance	Open MP profiler
#prefetches	Number of prefetched cache lines	SiGMA
Mpi_latesender	Time a receiving process is waiting for a message	Scalasca

© 2009 IBM Corporation

Bottleneck rule example

- a potential pipeline stalling problem caused by costly divide operations in a loop

#divides>0 && PM_STALL_FPU/PM_RUN_CYC>t && vectorized=0

  - #divides : number of divide operations
  - PM_STALL_FPU and PM_RUN_CYC: hardware counter events
  - t: threshold

© 2009 IBM Corporation

Metrics from the compiler

- Static analysis
  - Estimate of number of prefetchable streams
  - Estimate of pipeline stalls
  - Basic block information
- Optimization report


```
<Message>
<SourceId>1</SourceId><FileNumber>1</FileNumber>
<LineNumber>114</LineNumber><LoopId>6</LoopId>
<MessageId>131587</MessageId><SubKey>0</SubKey>
</Message>
```


© 2009 IBM Corporation

Solution Composition and Implementation

- Candidate solutions mined from expert knowledge
- Stored in the solution database
- Solutions are in generic forms and need to be instantiated. For example
  - Excessive time is spent on blocking MPI calls
  - To overlap computation with communication
  - Whether and how to overlap are application dependent

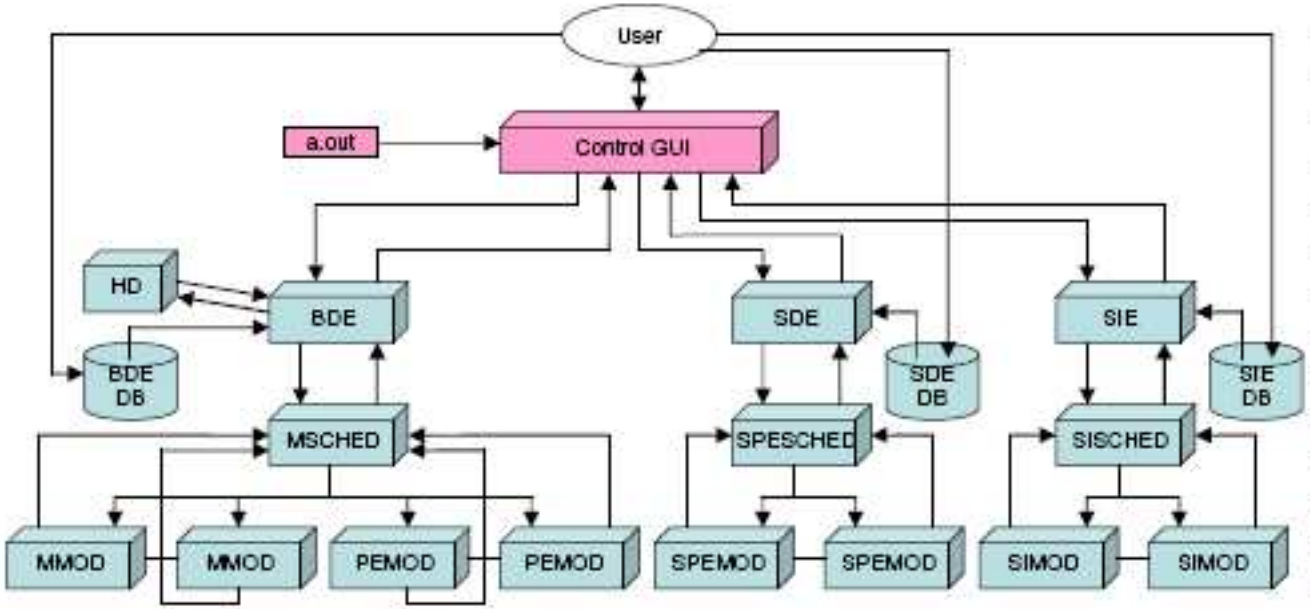
© 2009 IBM Corporation

Solution Composition and Implementation (continued)

- Solution determination/instantiation
  - Legality check
  - Parameter values computed
  - Performance improvement estimation
  - Code modification and environment setting determination
- Current solutions
  - Standard transformation through compiler
    - Compiler directives
    - Polyhedral framework
      - Customized optimization from standard transformation
  - Modifications to the source code
  - Suggestions

© 2009 IBM Corporation

Architecture of the Framework



Abbreviation	Meaning
BDE	Bottleneck Detection Engine
BDE DB	Bottleneck Detection Engine Database
HD	Hotspot Detector
MMOD	Metric Module
MSCHED	Metric Scheduler
PEMOD	Performance Estimation Module
SPEMOD	Solution Performance Estimation Module
SPESCHED	Solution Performance Estimation Scheduler
SDE	Solution Determination Engine
SDE DB	Solution Determination Engine Database
SIE	Solution Implementation Engine
SIE DB	Solution Implementation Engine Database
SISCHED	Solution Implementation Scheduler
SIMOD	Solution Implementation Module

© 2009 IBM Corporation

Case Study - LBMHD

- Lattice Boltzmann Magneto-Hydrodynamics code (LBMHD)
  - A mesoscopic description of the transport properties of physical systems using linearized Boltzmann equations.
  - Offers an efficient way to model turbulence and collisions in a fluid to model magneto-hydrodynamics
  - Performs a 2D simulation of high-temperature conduction

© 2009 IBM Corporation

Case Study – LBMHD (continue)

- Excessive stalls
- PM_CMPLU_STALL_LSU/PM_CYC > a and SA_STRIDE_ONE_ACCESS_RATE < b and SA_REGULAR_ACCESS_RATE(n) > SA_STRIDE_ONE_ACCESS_RATE + d
- if there is a significant number of cycles spent on LSU unit, and there are more n-stride accesses than stride-1 access, there is potentially a bottleneck

© 2009 IBM Corporation

Case Study – LBMHD (continue)

```
do j = jsta, jend
    do i = ista, iend
    ...
            do k = 1, 4
                    vt1 = vt1 + c(k,1)*f(i,j,k) + c(k+4,1)*f(i,j,k+4)
                    vt2 = vt2 + c(k,2)*f(i,j,k) + c(k+4,2)*f(i,j,k+4)
                    Bt1 = Bt1 + g(i,j,k,1) + g(i,j,k+4,1)
                    Bt2 = Bt2 + g(i,j,k,2) + g(i,j,k+4,2)
            enddo
    ...
            do k = 1, 8
    ...
                    feq(i,j,k)=vfac*f(i,j,k)+vtauinv*(temp1+trho*.25*vdotc+ &
                    .5*(trho*vdotc**2- Bdotc**2))
                    geq(i,j,k,1)= Bfac*g(i,j,k,1)+ Btauinv*.125*(theta*Bt1+ &
                    2.0*Bt1*vdotc- 2.0*vt1*Bdotc)
    ...
            enddo
    ...
    enddo
enddo
```

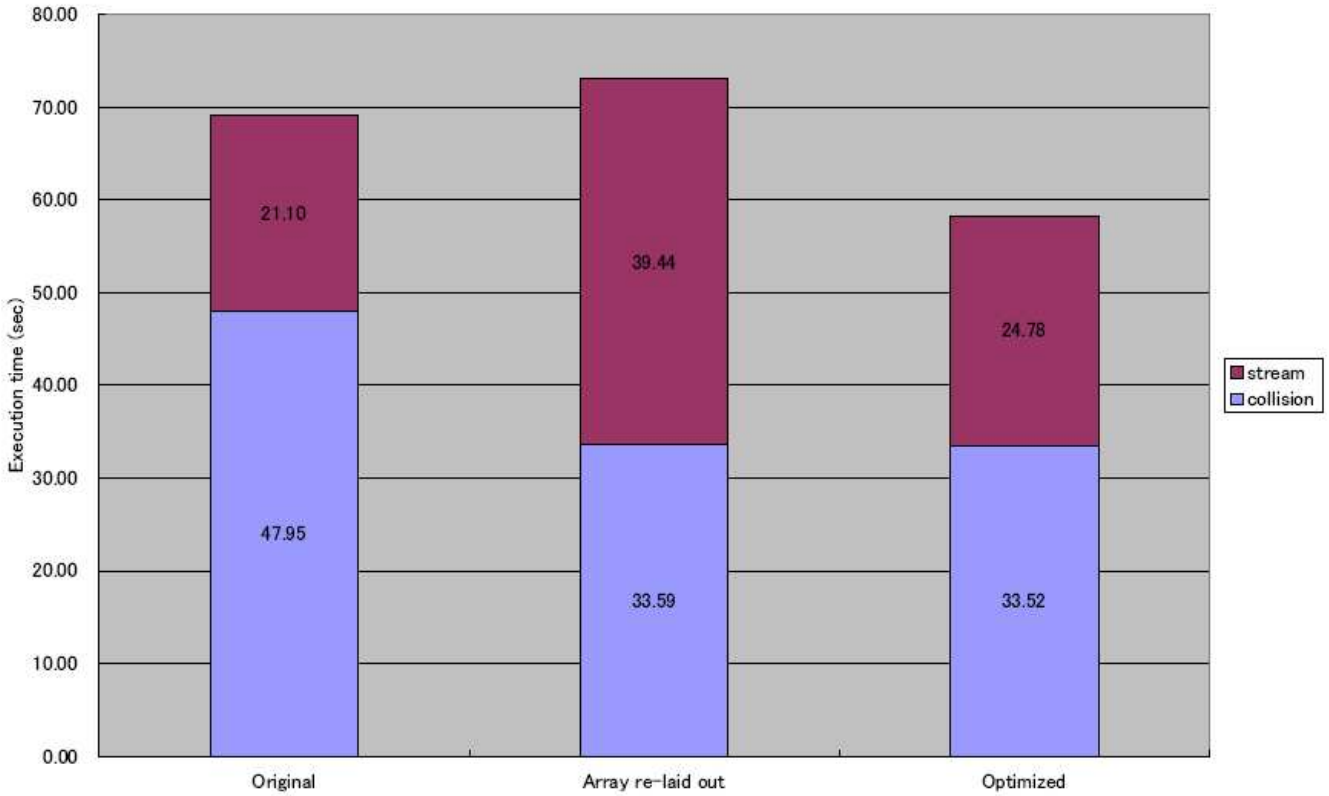
© 2009 IBM Corporation

Case Study – LBMHD (continue)

- For multi-dimensional arrays f, g, feq, and geq
  - The access order incurred by the j, i, k iteration order does not match with their storage order
  - Creates massive cache misses
- Two ways to match the array access order and the storage order
  - Change the access order by loop-interchange
    - Loops are not perfected nested
    - Impossible to implement loop interchange without violating the dependency constraints
  - Change the storage order to match the access order by re-laying out the array
    - Use compiler directives to implement the new storage order
    - !IBM SUBSCRIPTORDER(f(3, 1, 2), feq(3, 1, 2), g(4, 3, 1, 2), geq(4, 3, 1, 2))

© 2009 IBM Corporation

Case Study – LBMHD (continue)



20% improvement in execution time with a grid size 2048×2048 and 50 iterations on a P575+ (1.9 GHz Power5+, 16 CPUs. Memory: 64GB, DDR2) on one processor

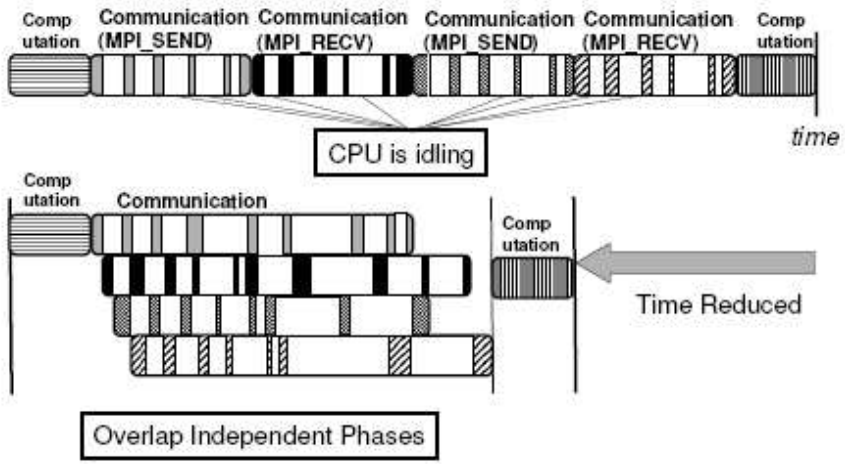
© 2009 IBM Corporation

Case Study – Distributed Poisson Solver

- Interleaved computation and communication phases
- All the communications in a phase are independent of each other, and can occur simultaneously

$$\frac{ElapsedTime - \frac{PM\_CYC}{CPUFrequency}}{ElapsedTime} > \alpha \text{ and } mpi\_hotspot = 1 \text{ and } \#blocking\_mpicalls > 0$$

- if the CPU spends a significant portion of its time idling in an MPI hotspot and there are blocking MPI calls, there is a potential bottleneck caused by the communication pattern.



© 2009 IBM Corporation

Case Study - Distributed Poisson Solver (continued)

- Solution
  - To initiate the communication as early as possible, and wait for its result as late as possible.
  - While the communication is taking place, more computation can be done
- Locations to place MPI calls
  - For each MPI call in the hotspot loop, generate lists of input (in) and output (out) variables.
  - Identify the first location, where the MPI call can be moved without breaking the original data dependency.
    - The earliest that a communication can be initiated.
  - Identify the last location where the MPI call can be moved to without breaking the original data dependency
    - The latest that a communication should complete.

© 2009 IBM Corporation

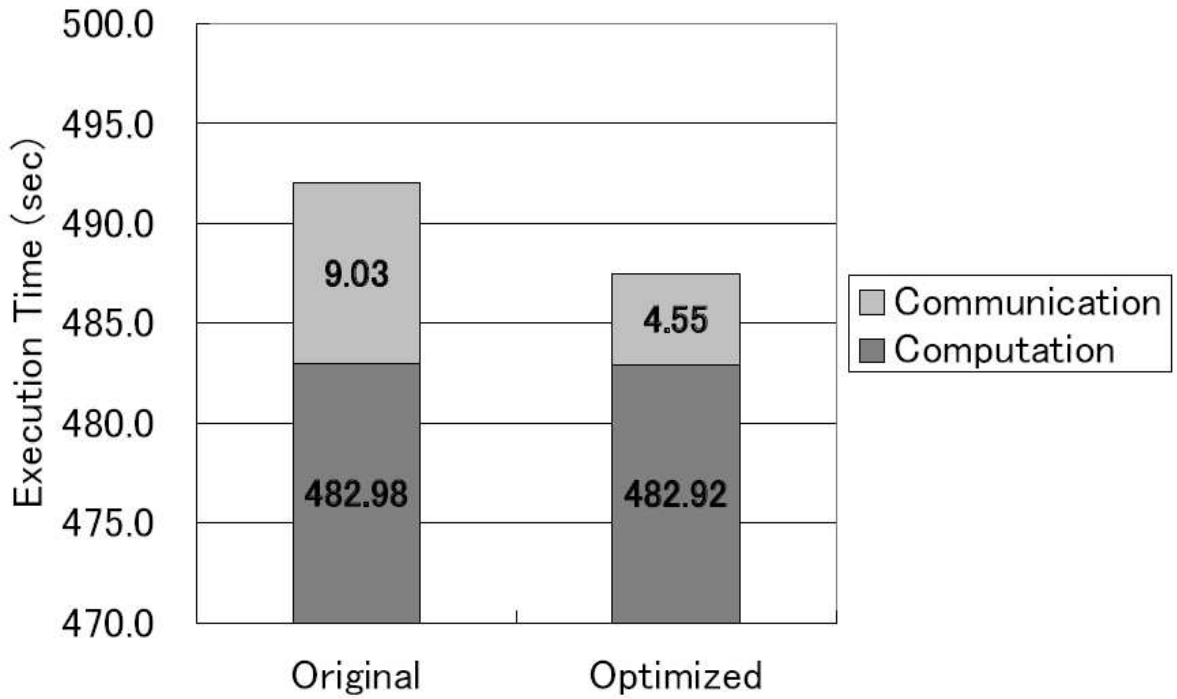
Case Study - Distributed Poisson Solver (continued)

- Rewrite MPI functions
- For example
- Original
  - call MPI_SEND(x, n, MPI_REAL, dst, 0, MPI_COMM_WORLD, istat, ierr)
- Modified
  - integer NEW0_1 ! Declaration
  - call MPI_ISEND(x, ..., NEW0_1, ierr) ! Initiation
  - call MPI_WAIT(NEW0_1, ..., ierr) ! Wait

© 2009 IBM Corporation

Case Study - Distributed Poisson Solver (continued)

- For a mesh size of 1G (1024 × 1024 × 1024), the optimization achieved about 50% improvement in communication time on Blue Gene/P



© 2009 IBM Corporation

Conclusion and Future Work

- High productivity performance tuning
  - Unifying performance tools, compiler, and expert knowledge
  - Metrics from performance data collected by existing performance tools
  - The analysis of multiple tools can be correlated and combined through bottleneck rules.
- Future work
  - Populate the databases with more rules and solutions

© 2009 IBM Corporation

© 2009 IBM Corporation

HD Results (Loop Level)

hpcs_gui

File Action Database Window

BDE | BDE and SDE

DATA WINDOW

HD | BDE

	SELECTION	FILE	FUNCTION	START	END L	self secs (P0)
1	☑ Loop	mhd.F	collision	512	592	5.03
2	☐ Function	mhd.F	collision	513	596	5.03
3	☐ Function	mhd.F	stream	338	494	3.92
4	☑ Loop	mhd.F	stream	416	493	2.67
5	☐ Loop	mhd.F	stream	339	365	0.64
6	☐ Loop	mhd.F	stream	367	414	0.61
7	☑ Function	mhd.F	neighbours	619	807	0.22
8	☑ Loop	mhd.F	neighbours	750	804	0.14
9	☑ Loop	mhd.F	equil	291	314	0.13
10	☐ Function	mhd.F	equil	292	317	0.13
11	☐ Loop	mhd.F	check	927	933	0.07
12	☐ Function	mhd.F	check	924	934	0.07
13	☐ Loop	mhd.F	neighbours	693	748	0.03
14	☐ Loop	mhd.F	neighbours	618	626	0.03
15	☐ Function	mhd.F	init	237	265	0.02
16	☐ Loop	mhd.F	neighbours	644	651	0.02
17	☐ Loop	mhd.F	init	273	281	0.02
18	☐ Loop	mhd.F	neighbours	681	689	0
19	☐ Function	mhd.F	my_range	224	226	0
20	☐ Loop	mhd.F	init	264	267	0
21	☐ Loop	mhd.F	init	268	271	0
22	☐ Function	mhd.F	mhd	31	145	0
23	☐ Loop	mhd.F	neighbours	656	664	0
24	☐ Loop	mhd.F	mhd	117	129	0
25	☐ Function	mhd.F	decomp	157	214	0
26	☐ Loop	mhd.F	neighbours	899	906	0

SOURCE WINDOW

mhd.F

```
421     !fetch one line strides = 16 elements at a time
422     ptr => geq(inmp-1, j, 1, k)
423     CALL Mambo_Prefetch(ptr,  128, 1, NMP_PREF_LINES)
424     ptr => geq(inmp, j-1, 3, k)
425     CALL Mambo_Prefetch(ptr,  128, 1, NMP_PREF_LINES)
426     ptr => geq(inmp+1, j, 5, k)
427     CALL Mambo_Prefetch(ptr,  128, 1, NMP_PREF_LINES)
428     ptr => geq(inmp, j+1, 7, k)
429     CALL Mambo_Prefetch(ptr,  128, 1, NMP_PREF_LINES)
430     do i = inmp, MIN(inmp+16*NMP_PREF_LINES-1, iend)
431     #else
432     do i = ista, iend
433     #endif
434       g(i,j,1,k)= geq(i-1,j,1,k)
435       g(i,j,3,k)= geq(i,j-1,3,k)
436       g(i,j,5,k)= geq(i+1,j,5,k)
437       g(i,j,7,k)= geq(i,j+1,7,k)
438     enddo
439     #ifdef NMP
440     enddo
441     #endif
442     enddo
443
444
445     do j = jsta, jend
```

© 2009 IBM Corporation

Source Code with Clock Ticks

hpcs_gui

File Action Database Window

BDE | BDE and SDE

mhd.F

```
347          ptr => feq(inmp, j-1, 3)
348          CALL Mambo_Prefetch(ptr,   128, 1, NMP_PREF_LINES)
349          ptr => feq(inmp+1, j, 5)
350          CALL Mambo_Prefetch(ptr,   128, 1, NMP_PREF_LINES)
351          ptr => feq(inmp, j+1, 7)
352          CALL Mambo_Prefetch(ptr,   128, 1, NMP_PREF_LINES)
353          do i = inmp, MIN(inmp+(16*NMP_PREF_LINES)-1, iend)
354          #else
355          do i = ista, iend
356          #endif
357          !  write(*,*) i, ' ', j, ' ', inmp
358     21     f(i,j,1)= feq(i-1,j,1)
359     28     f(i,j,3)= feq(i,j-1,3)
360      9    f(i,j,5)= feq(i+1,j,5)
361      6    f(i,j,7)= feq(i,j+1,7)
362          enddo
363          #ifdef NMP
364          enddo
365          #endif
366          enddo
367
368          do j = jsta, jend
369          #ifdef NMP
370          do inmp = ista, iend-16*NMP_PREF_LINES+1, 16*NMP_PREF_LINES
371          !fetch one line strides = 16 elements at a time
372          ptr => feq(inmp, j, 2)
373          CALL Mambo_Prefetch(ptr,   128, 1, NMP_PREF_LINES)
```

© 2009 IBM Corporation

BDE Results

hpcs_gui

File Action Database Window

SDE

HD | BDE

	BOTTLENEC	DIMENS	DESCRIPTION	RULE	FUNCTION	START LI	END LINE
1	DcacheMiss	CPU	cycles wasted due to stalls	(PM_RUN_CYC - (PM_GCT_EMPTY_CYC + PM_GRP_CMPL))	stream	416	493
2	DcacheMiss	CPU	cycles wasted due to stalls	(PM_RUN_CYC - (PM_GCT_EMPTY_CYC + PM_GRP_CMPL))	equil	291	314
3	DcacheMiss	CPU	cycles wasted due to stalls	(PM_RUN_CYC - (PM_GCT_EMPTY_CYC + PM_GRP_CMPL))	neighbours	750	804
4	DcacheMiss	CPU	cycles wasted due to stalls	(PM_RUN_CYC - (PM_GCT_EMPTY_CYC + PM_GRP_CMPL))	collision	512	592
5	DcacheMiss	CPU	cycles wasted due to stalls	(PM_RUN_CYC - (PM_GCT_EMPTY_CYC + PM_GRP_CMPL))	neighbours	619	807
6	MPIBarrier1	COMMU	imbalanced wait time	MAX(mpi_wait_hot_sum_time)-MIN(mpi_wait_hot_sum_time) > 1.	neighbours	619	807
7	Stalls	CPU	cycles spent in stalls	(PM_CMPLU_STALL_LSU + PM_CMPLU_STALL_FXU + PM_CM	neighbours	619	807
8	Stalls	CPU	cycles spent in stalls	(PM_CMPLU_STALL_LSU + PM_CMPLU_STALL_FXU + PM_CM	stream	416	493
9	Stalls	CPU	cycles spent in stalls	(PM_CMPLU_STALL_LSU + PM_CMPLU_STALL_FXU + PM_CM	collision	512	592
10	Stalls	CPU	cycles spent in stalls	(PM_CMPLU_STALL_LSU + PM_CMPLU_STALL_FXU + PM_CM	neighbours	750	804
11	Stalls	CPU	cycles spent in stalls	(PM_CMPLU_STALL_LSU + PM_CMPLU_STALL_FXU + PM_CM	equil	291	314

© 2009 IBM Corporation

Query Execution

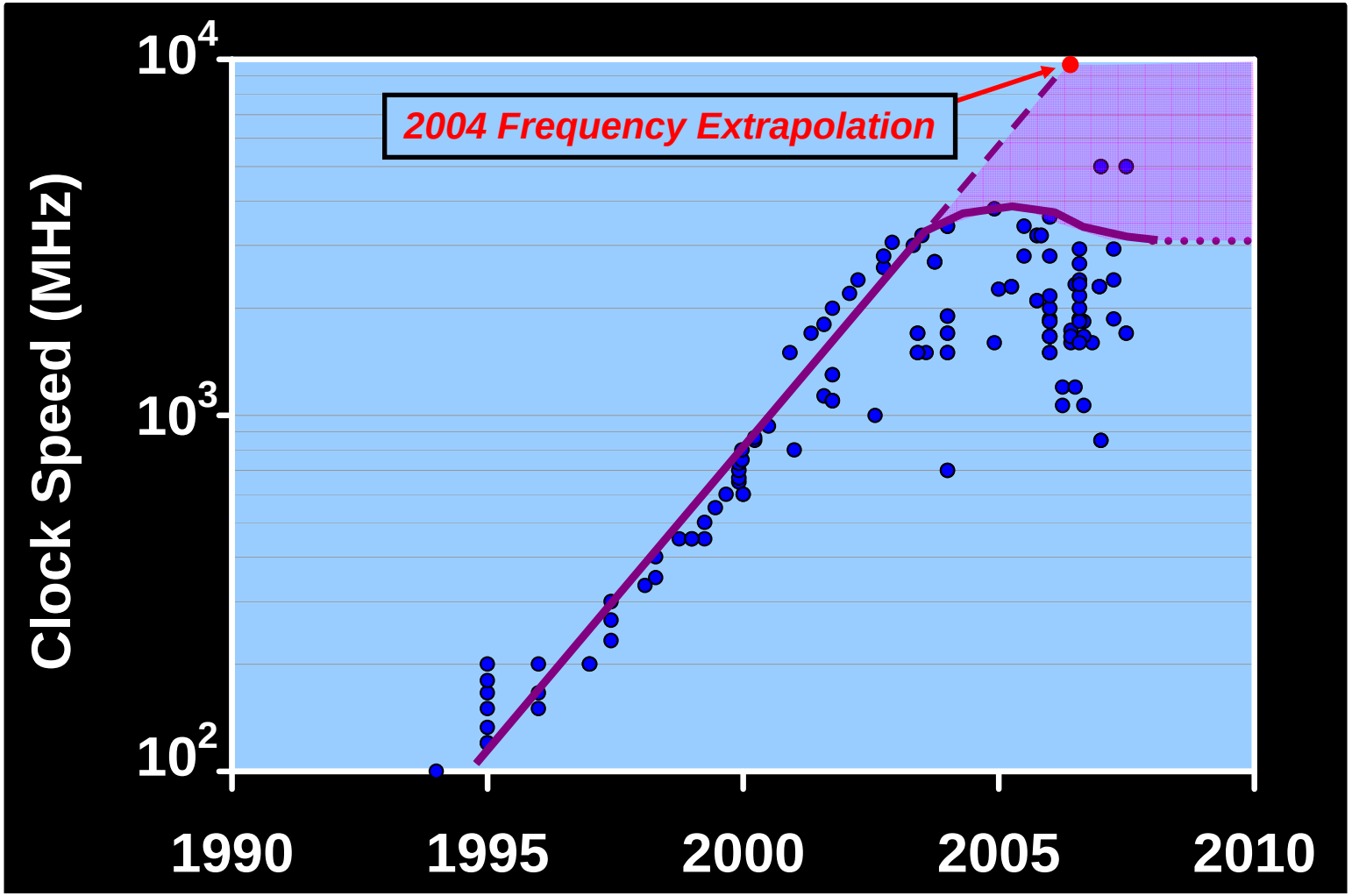
hpcs_gui <2>

Label	metric	Value for P0	MIN	MAX	AVG
Fri Apr 11 15:36:43 2008					
/gsa/yktgsa-h1/01/hfwen/test/DARPA/mhd.AIX/mhdNopg					
mpi					
mhd.F:collision:512:592					
mhd.F:equil:291:314					
mhd.F:neighbours:619:807					
	mpi_all_hot_avg_time	0.000772	0.000772	0.000954	0.000439
	mpi_all_hot_max_time	0.079773	0.079773	0.078625	0.042089
	mpi_all_hot_med_time	0.000005	0.000005	0.000005	0.000007
	mpi_all_hot_min_time	0.000001	0.000001	0.000001	0.000001
	mpi_all_hot_sum_time	3.089104	3.089104	3.816296	1.757381
	mpi_all_hot_var_time	0.000055	0.000055	0.000068	0.000031
	mpi_all_prog_avg_time	0.000955	0.000955	0.000946	0.000483
	mpi_all_prog_max_time	0.079773	0.079773	0.078625	0.042089
	mpi_all_prog_med_time	0.000005	0.000005	0.000005	0.000007
	mpi_all_prog_min_time	0.000001	0.000001	0.000001	0.000001
	mpi_all_prog_sum_time	3.854623	3.854623	3.817104	1.949907
	mpi_all_prog_var_time	0.000069	0.000069	0.000067	0.000034
	mpi_wait_hot_avg_time	0.001535	0.001535	0.001900	0.000864
	mpi_wait_hot_max_time	0.079773	0.079773	0.078625	0.042089
	mpi_wait_hot_med_time	0.000002	0.000002	0.000003	0.000003
	mpi_wait_hot_min_time	0.000001	0.000001	0.000001	0.000001
	mpi_wait_hot_sum_time	3.069639	3.069639	3.800613	1.727596
	mpi_wait_hot_var_time	0.000110	0.000110	0.000134	0.000061
	mpi_wait_prog_avg_time	0.001535	0.001535	0.001900	0.000864
	mpi_wait_prog_max_time	0.079773	0.079773	0.078625	0.042089
	mpi_wait_prog_med_time	0.000002	0.000002	0.000003	0.000003
	mpi_wait_prog_min_time	0.000001	0.000001	0.000001	0.000001
	mpi_wait_prog_sum_time	3.069639	3.069639	3.800613	1.727596
	mpi_wait_prog_var_time	0.000110	0.000110	0.000134	0.000061
mhd.F:neighbours:750:804					
mhd.F:stream:416:493					
pomprof					
Mon Apr 7 11:27:04 2008					

© 2009 IBM Corporation

Microprocessor Clock Speed Trends

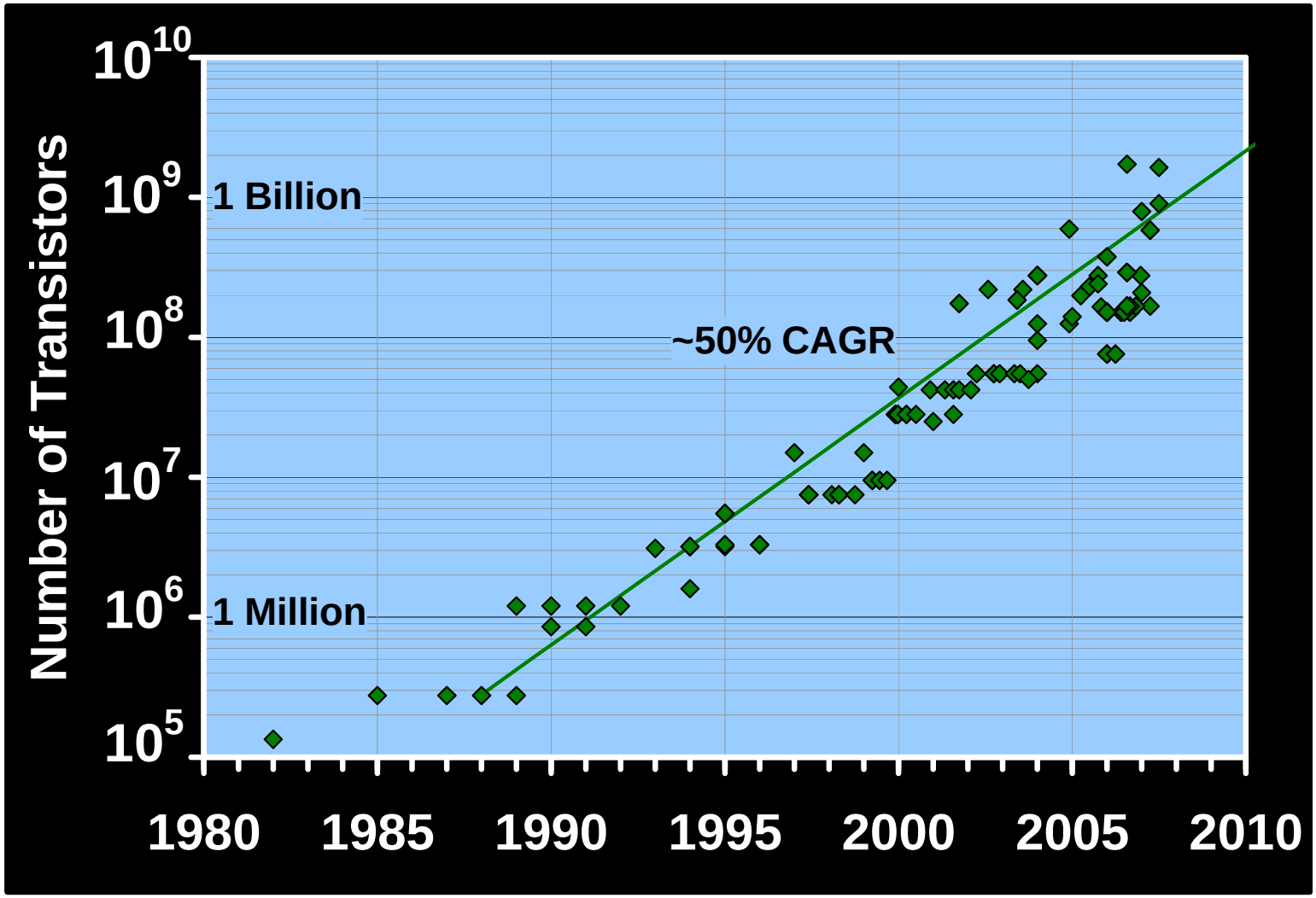
Managing power dissipation is limiting clock speed increases



© 2009 IBM Corporation

Microprocessor Transistor Trend

Lithography will continue to deliver density scaling



© 2009 IBM Corporation

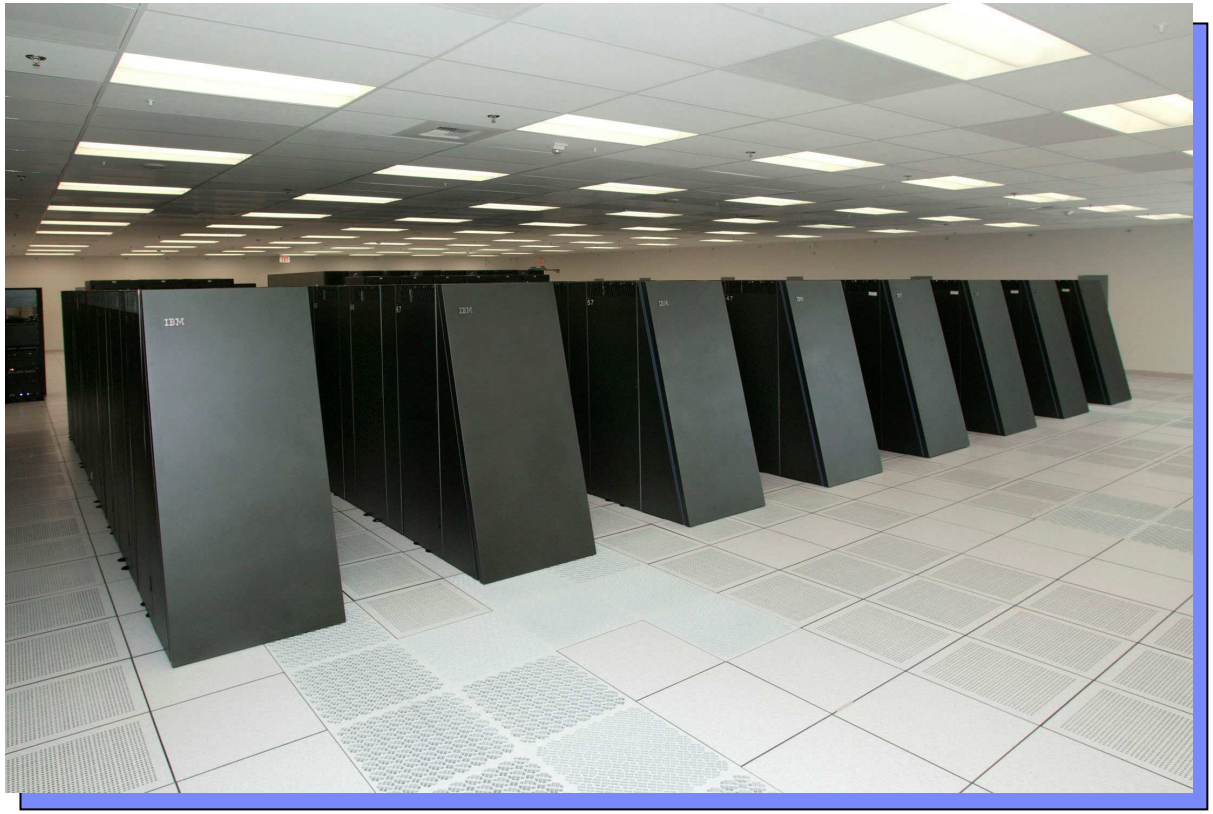
Hardware trends that address the power problem

- Observation
  - **Although frequency scaling is "dead", Moore's Law is still alive: transistor density continues to increase exponentially**
- Trend #1: Multi-core processor chips
  - **Maintain (or even reduce) frequency while replicating cores**
- Trend #2: Accelerators
  - **Previously, processors would "catch" up with accelerator function in the next generation**
    - **Accelerator design expense not amortized well**
  - **New accelerator designs will maintain their speed advantage**
  - **And will continue an enormous power advantage for target workloads**

© 2009 IBM Corporation

Blue Gene/P, an example of addressing power in a massive scale-out system

- 40K compute processors
  - **557 Teraflop Peak**
- 80 Terabytes memory
- 3D torus interconnect
- Collective and barrier networks
- Power:
  - **0.33 Gigaflop/W**
- **40 compute racks**



BG/P at ANL, #4 on the Top500 list

© 2009 IBM Corporation

IBM Roadrunner – a system with accelerators

- **Architecture**
  - 12,960 IBM PowerXCell 8i CPUs
  - 6,480 AMD Opteron dual-core processors
  - Infiniband, Linux
- **Power**2.35 MW
- **Space**296 racks, 6,000 sq ft (560 m^2)
- **Memory**103.6 TiB
- **Speed**1.7 petaflops (peak)

Roadrunner at LANL, #1 on the Top500 list

© 2009 IBM Corporation