Programming Models For Petascale Era

Jarek Nieplocha
Laboratory Fellow, Chief Scientist
Computational Science and Mathematics
Pacific Northwest National Laboratory

Washington State University

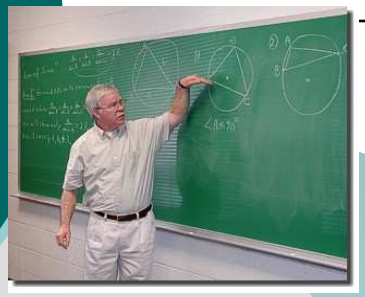
Focus of this talk

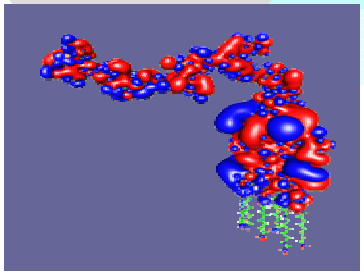
- Successful programming models will help address challenges of petascale computing
- To address the issue we will look at
  - Hardware trends
  - Application characteristics
  - Programming models and Runtime
  - Promising ideas and technologies

Software for Petascale Systems

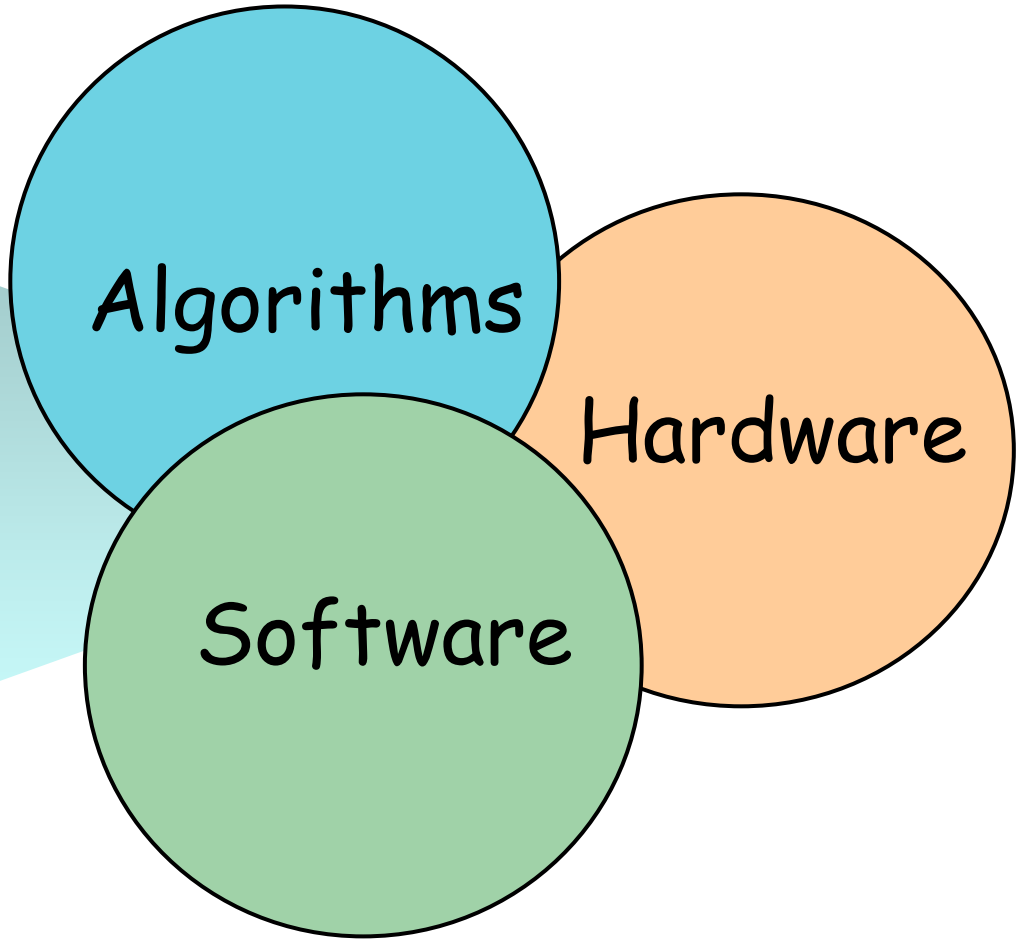
- To develop applications for solving grand challenge problems on petascale systems teams are required across different areas
  - CS, Math
  - Domain science
  - Understanding of h/w and OS
  - Team efforts
- The current leadership computing investments are in HARDWARE and not balanced in SOFTWARE efforts
- System and application software activities will require comparable or larger investments

Exploiting HPC architectures is becoming harder because of size and complexity of the systems and applications themselves





Success relies on coupling multiple areas of science and technology



There are many dependencies….

Hardware
- system arch.
- processor arch.
- h/w accelerators
- memory b/w
- interconnect b/w
- secondary storage
- reliability

Software
- scalable OS
- programming model
- portability
- communication libs
- numerical libraries
- compilers
- debuggers

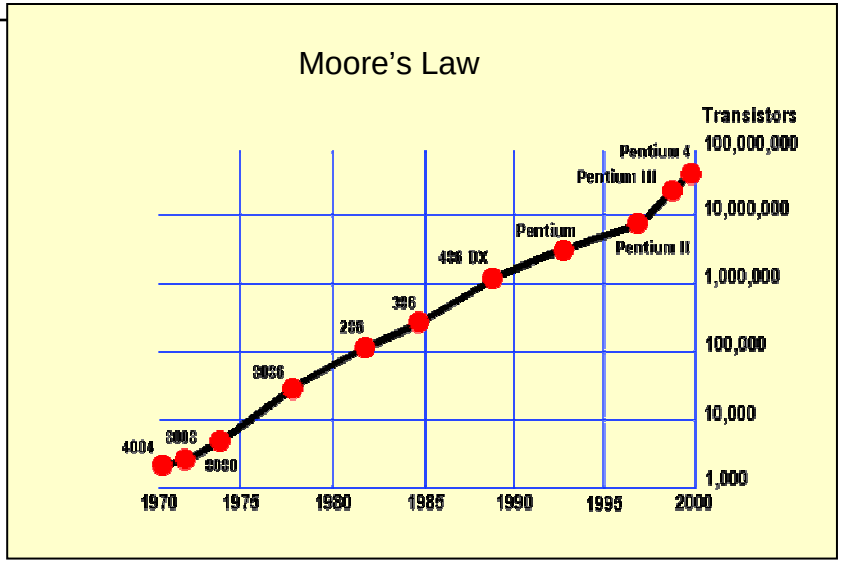
Algorithms
- scalable
- resource efficient
- comp. complexity
- data decomposition
- locality space/time
- load balancing
- multiple levels of parallelism

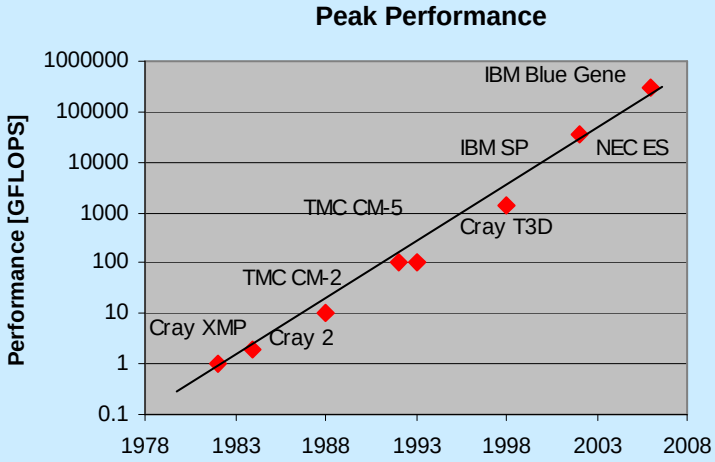
Key Challenges in Petascale Computing and Beyond

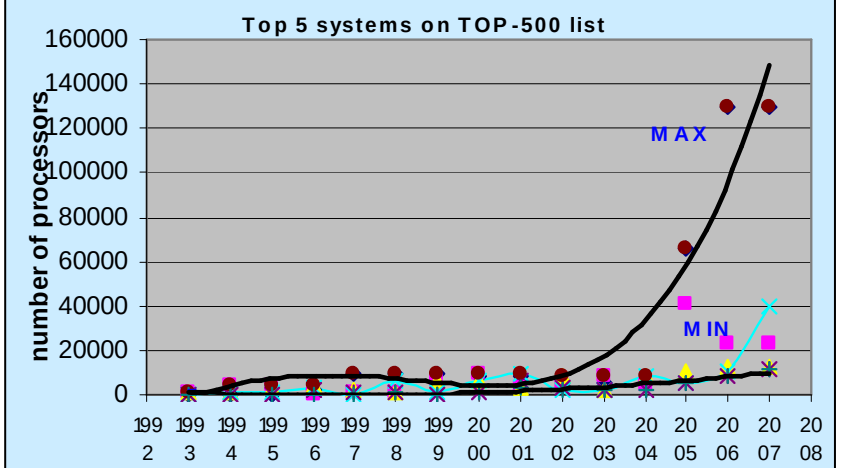
Scalability	How can we adopt applications, algorithms and system software to massive processor configurations?
Processor Performance	Single processor (socket) is now multicore and becoming heterogenous. Memory b/w shortage.
Fault Tolerance	Massive number of components leads to more frequent hardware faults
Productivity	Development of scalable, reliable, and efficient applications is becoming harder and more costly
Power Management	The cost of 20-30MW power for petascale systems is already prohibitive

Petascale Implies Massive Parallelism

- Computational speed in last *two decades*
  - 10,000 for single processor
    - One year vs one hour time to solution
  - 40,000 for supercomputers
    - Cray XMP in 1982 vs IBM BGL now
- Parallelism in leadership systems
  - 4 CPU in Cray XMP in 1982
  - 130,000 in the IBM BG/L in 2006

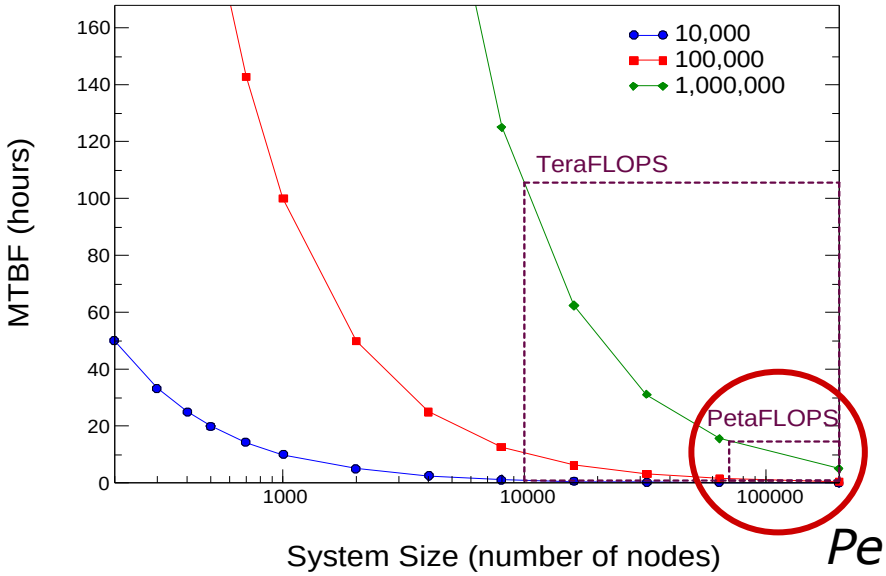






Implications of Massive Parallelism

Mean Time Between Failure (MTBF) as a function of system size

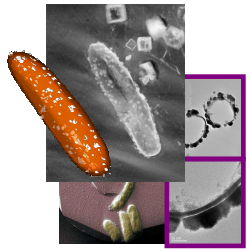


- Component count in high-end systems has been growing
- How do we utilize large (10^5 processor) systems for solving complex science problems?
- Successful programming models must help address two challenges
  - Scalability to massive processor counts
  - Hardware and software failures

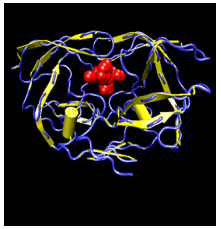
Petascale+

Applications and Fault Tolerance

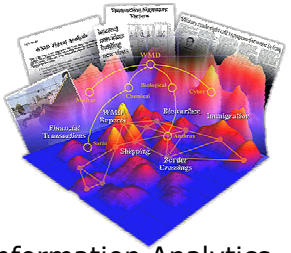
- Significant variability in application characteristics
  - Multidisciplinary, multiresolution, and multiscale nature
  - Increasingly differing demands on the system resources: disk, network, memory, CPU usage
  - Some of them have natural fault resiliency and require very little support
- System Factors
  - Different I/O configurations, programmable or simple/commodity NICs, proprietary/custom/commodity operating systems
- Tradeoffs between acceptable failure rates & cost
  - Cost effectiveness is the main constraint in HPC
  - Therefore, it is not cost-effective or practical to rely on a single fault tolerance approach for all applications and systems



Computational Biology
dynamic task model
shared database

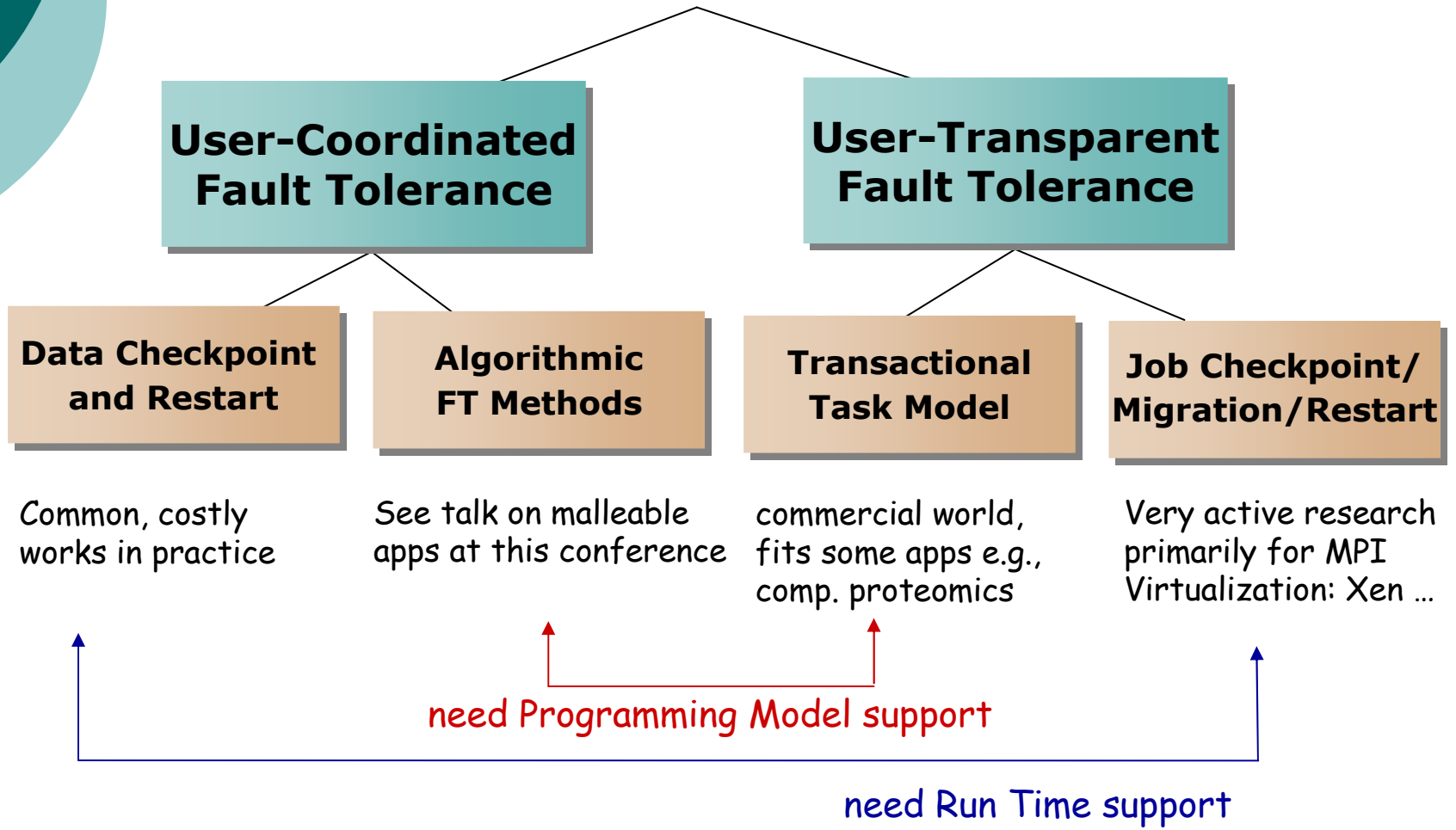


Molecular Dynamics
state easily recomputed



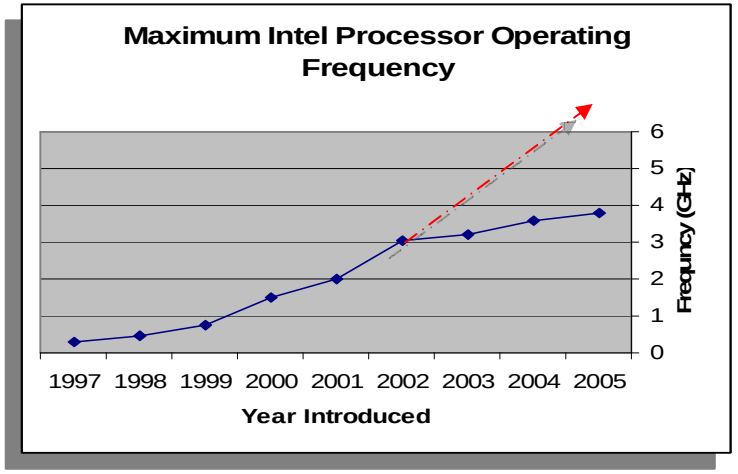
Information Analytics
collective comms

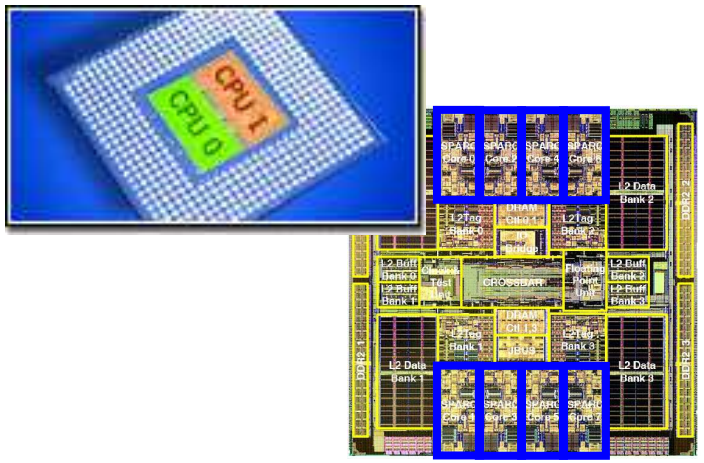
Programming Models and FT methods



Parallel Processing Enters Mainstream

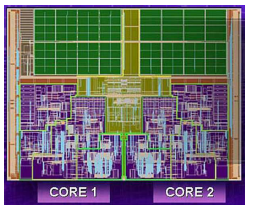
- CMOS technology clock freq. limits due to the power dissipation (<10GHz)
- Moore Law gives us multiple-core processors
  - Two-, four-, eight- cores now
  - 16- up to 128- cores discussed
- Parallel processing became the primary technique for accelerating performance on commodity computers



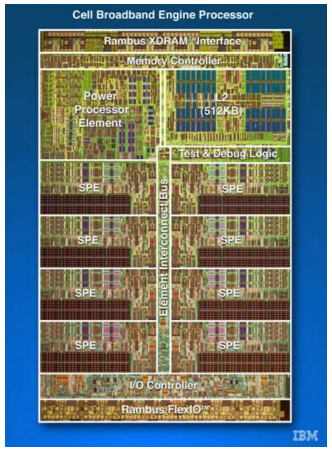


8-core Sun Niagara

From Multi- to Many-Core



- Multi-core (2-4 cores) designs dominate the commodity market and percolate into high-end systems
- Many-core (10s or 100+ cores) is emerging
  - heterogenuity is a real possibility
- Examples
  - Intel 80-core TeraScale chip & Larrabee chip
  - IBM Cyclops-64 chip with 160 thread units
  - ClearSpeed 96-core CSX chip
  - NvidiaTesla products based on 128-core C870 GPU (0.5TFLOP)

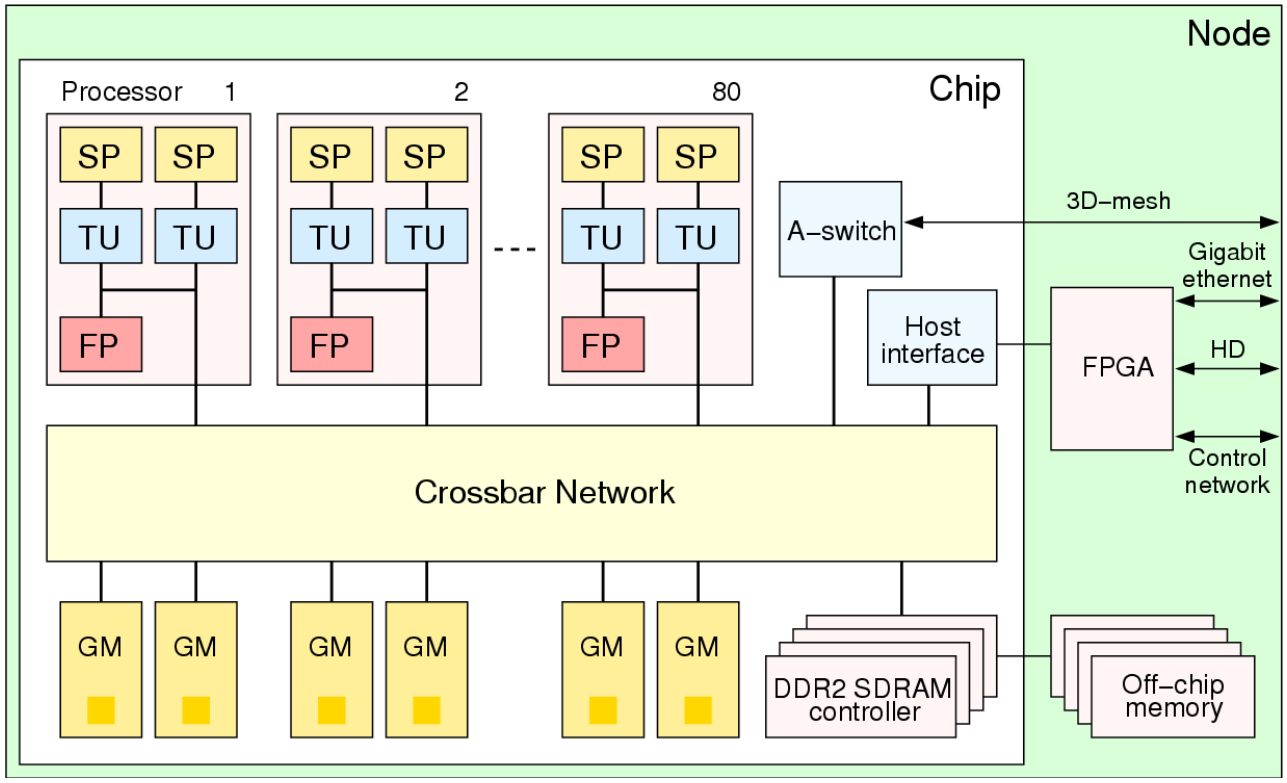


IBM Cell

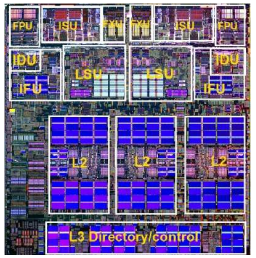
Future Multicore Systems Might Look Like the IBM Cyclops

IBM Cyclops-64 Chip Architecture

- 160 hardware thread units
- three-level explicit memory hierarchy
- thread execution support



Implications of Multicores



- To achieve performance on multi and many core systems for a single socket we rely on techniques reminiscent of traditional HPC
  - Exploit concurrency at the algorithmic level
  - Design efficient parallel algorithms
  - Memory bandwidth (bytes/flop) is very limited and cannot be wasted
    - Number of wires: IO is the true bottleneck
    - "flops are free, bandwidth is expensive"
  - Minimize inter-processor/core communication and synchronization

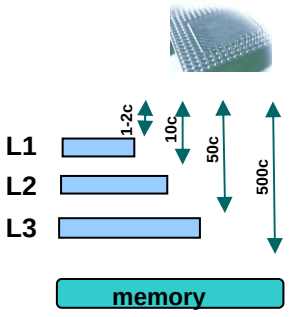
System Balance in Three Machine Generations

Examples from PNNL

	IBM SP	HP/QSNet2	Dell 1950/IBA
Year	1996	2003	2006
Processor	IBM P2SC	IA-64	Intel Woodcrest
Cost including network	$40K/node	$20K/node	$6K/node
Number of processors/node	1	2	2 dual cores
Peak processor performance (Gflops)	0.48	6.0	12 x 2
Peak node performance (Gflops)	0.48	12	48
Cache size	128KB	1MB	4MB
Memory B/F (bytes /flops)	4.2	0.75	0.4
Network B/F (unidirectional)	0.17	0.075	0.027
MPI latency (microsec)	40	2.5	3

Addressing System Imbalance

- Special efforts are needed to address performance constraints of the current technology
- Memory bandwidth is a big issue
  - Emerging multicore systems have less b/w available than past Uni-processor and then SMP designs
  - Both for local and remote memory
- How to reduce bandwidth usage
  - Avoid memory copies, packing/unpacking, etc
  - Smart data placement to reduce cost of data movement
  - Shared memory within the node (cores and sockets)
- Hiding cost of data transfer across the network
  - Prefetching, overlapping communication with computations
  - Some communication models more effective than others

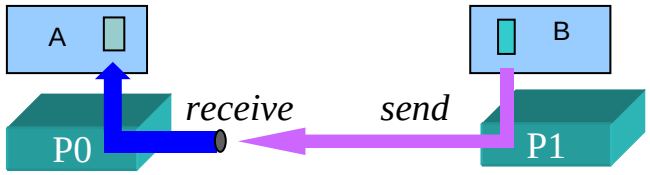


Interprocessor Communication

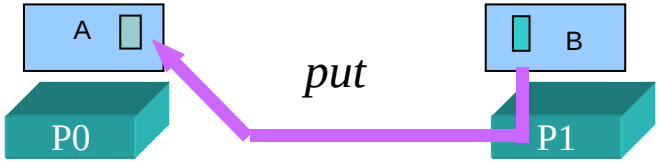
- How to transfer data between processors?
  - Can be a key scalability bottleneck
  - 1-sided models better at latency hiding
- Communication models assume certain hardware model
- Message-Passing (MPI) is based on the abstract machine model of 1980 hardware
  - uniprocessor nodes
  - no virtual or shared memory
  - no Remote Direct Memory Access (RDMA) h/w capabilities

Communication Models

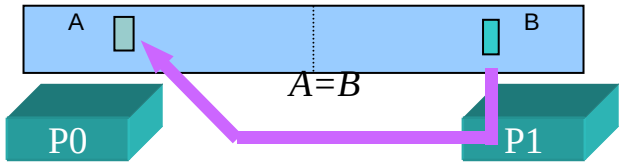
Example: Copy B on P1 to A on P0



message passing
2-sided model



remote memory access (RMA)
1-sided model

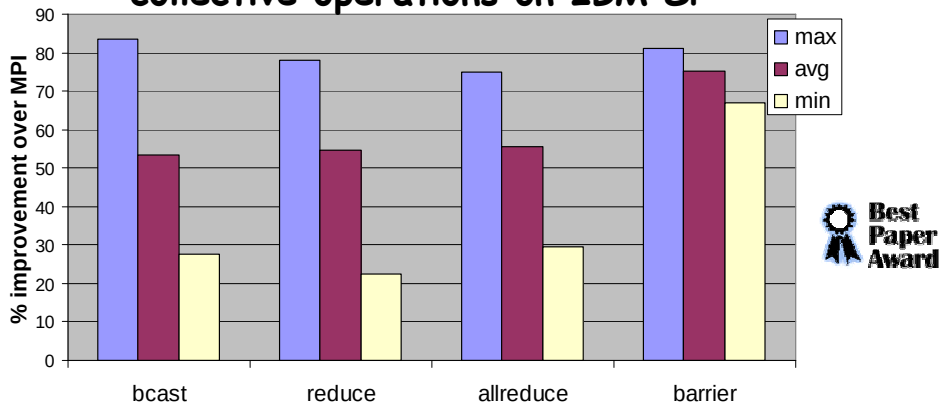


shared memory load/stores
0-sided model

Does MPI guarantee we get most efficient performance on modern hardware?

- Many believe it is not anymore
- Experiment: replace MPI with hardware native protocols
  - Shared memory within SMP node
  - RMA/RDMA between nodes
  - Overlapping computations and comms
- Examples
  - Dense parallel matrix multiplication
  - Collective operations

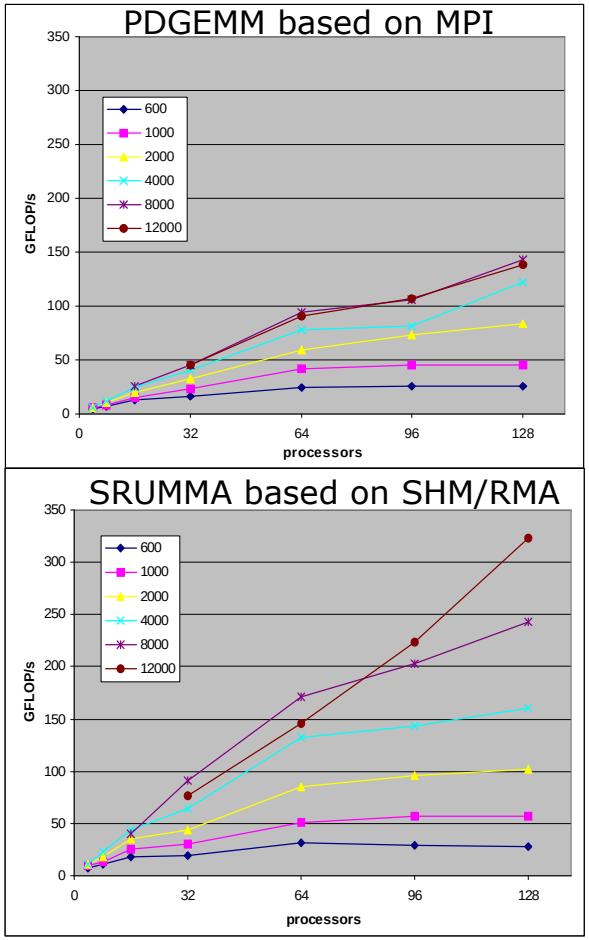
collective operations on IBM SP



Best Paper Award

see paper at IPDPS'03 by Tipparaju, Nieplocha, Panda

dense matrix multiplication aggregate performance



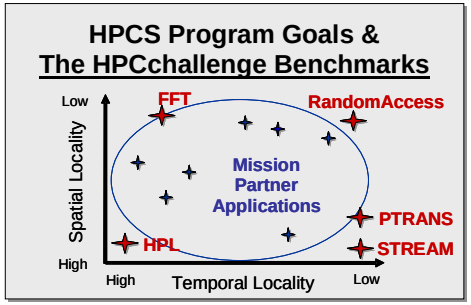
SRUMMA paper at IPDPS'04

Performance and Productivity

- Both determine the total cost of solution
- Focus on performance is obvious
  - how effectively can we use the h/w?
- Productivity as a part of the cost
  - the dramatic improvement in h/w cost effectiveness had limited effect on improving programmer productivity
- DARPA High Productivity Computing is pursuing a holistic approach
  - Hardware
  - System software
  - Novel programming languages: Cray Chapel, IBM X10, Sun Fortress
  - Requires vendors to analyze and present effectiveness for real science applications

HPCS

Program recognizes software and hardware must work together in integrated ways to achieve programmer productivity

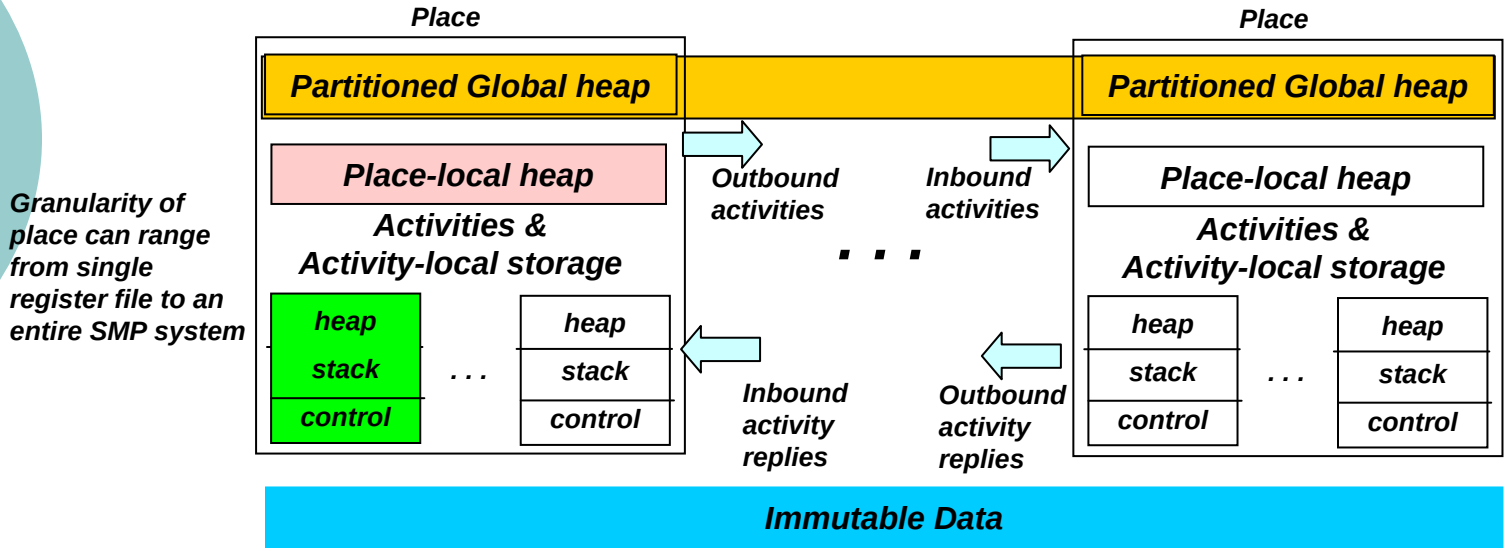


Parallel Programming Models

- How do we program the parallel machine?
- Single Execution Stream
  - Data Parallel, e.g. HPF
- Multiple Execution Streams
  - Partitioned-Local Data Access
    - MPI
  - Uniform-Global-Shared Data Access
    - OpenMP
  - Partitioned-Global-Shared Data Access
    - Co-Array Fortran
  - Uniform-Global-Shared + Partitioned Data Access
    - UPC, Global Arrays, X10

GAS models

Example HPCS Languages: IBM X10

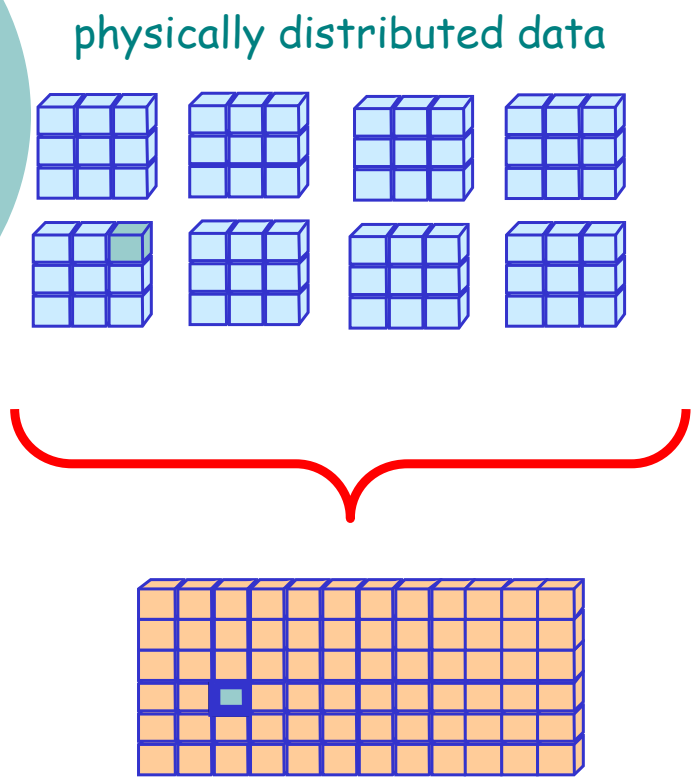


Granularity of place can range from single register file to an entire SMP system

- **A program is a collection of *places*, each containing *resident data* and a *dynamic* collection of *activities*.**
- **Program may *distribute* aggregate data (arrays) across *places* during allocation.**
- **Program may directly operate only on local data, using *atomic blocks*.**
- **Program may *spawn* multiple (local or remote) activities in parallel.**
- **Program must use *asynchronous operations* to access/update remote data.**
- **Program may *detect termination* or *(repeatedly) detect quiescence* of a data-dependent, distributed set of activities.**

Credit: Vijay Saraswat, IBM

Global Arrays Toolkit delivers global address space abstractions as a library

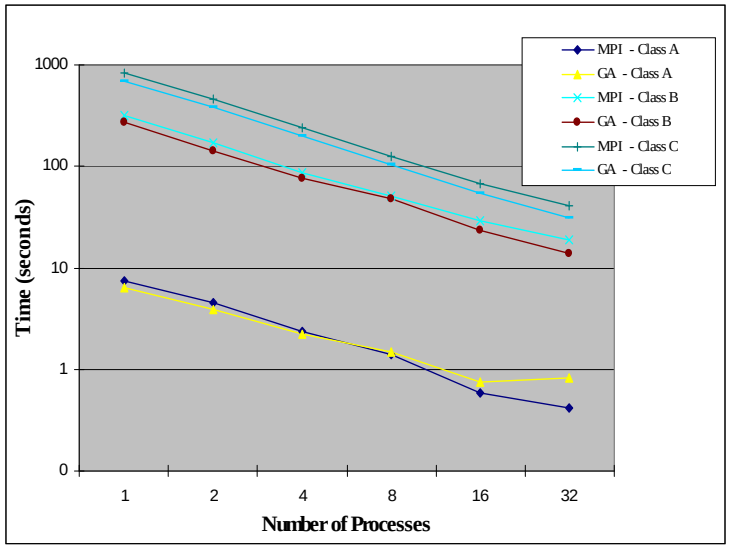


- Works with MPI
- Multilanguage bindings
- Implemented using ARMCI one-sided communication library
  - Optimized for shared memory and RDMA protocols
  - Zero-copy communication
- GA is the most mature alternative to MPI and OpenMP standard programming models
  - Multiple application areas
  - Production use
- Extensive functionality: >150 calls
- Logically shared view of distributed data, yet data locality control similar to MPI model
  - Every piece of shared data has a logical home
  - User can control and exploit locality when needed, ignore in other cases

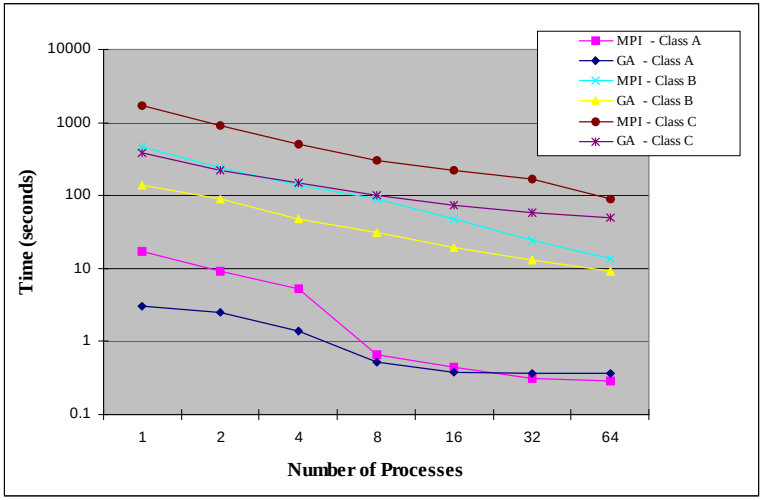
NAS CG Benchmark

- MPI implementation is very good and hard to outperform
  - ZPL, OpenMP, CAF, HPF
- GA implementation exploits:
  - Locality+ machine topology
  - shared memory (direct access load/stores)
  - nonblocking RDMA

Linux Cluster



SGI Altix



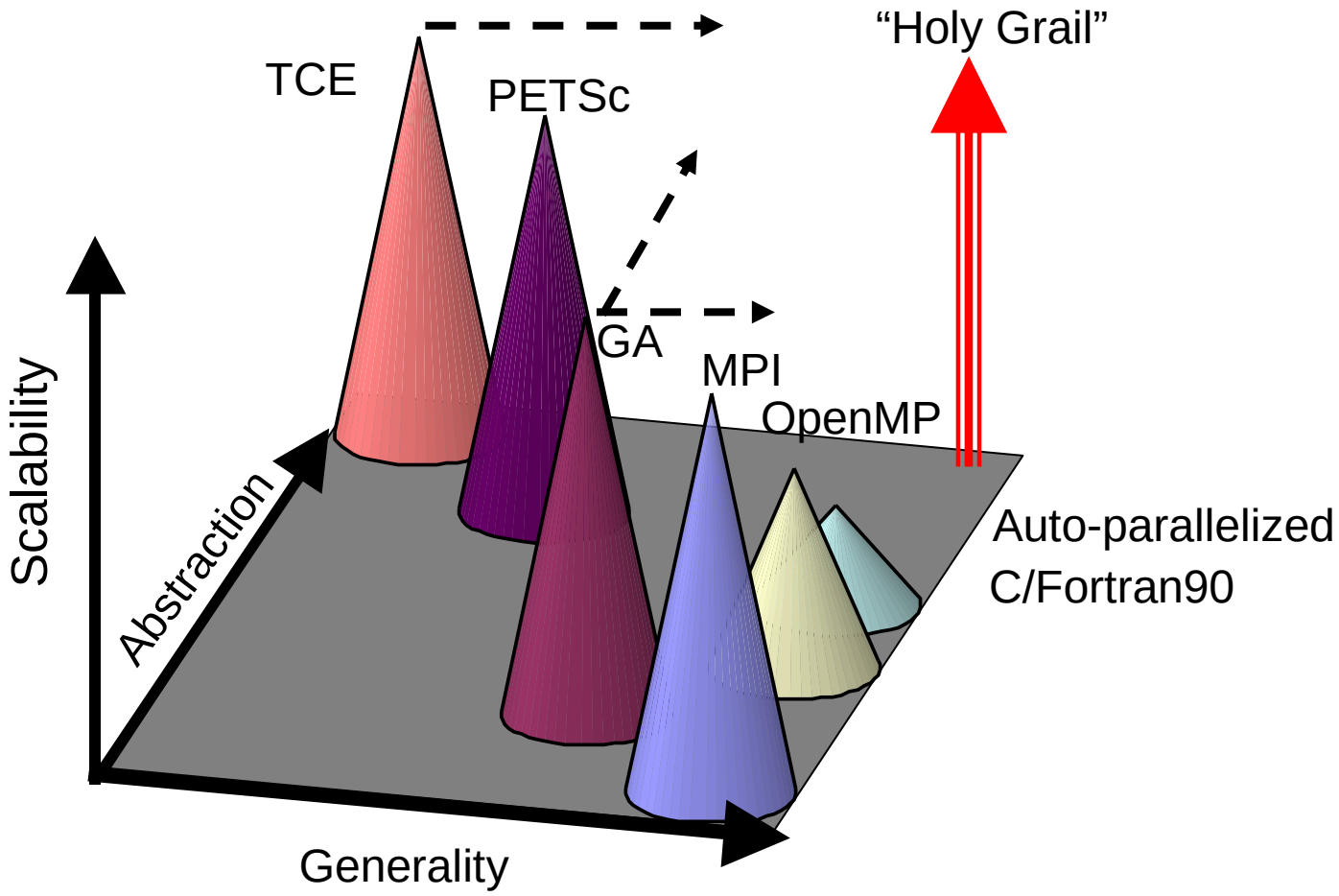
Characteristics of Programming Models

	Shared memory	Message Passing 2-sided	Message passing 1-sided	Global Address Space
Data view	shared	distributed	distributed	distributed or shared
Access to data	simplest (*a=b*)	hard (*send-receive*)	moderate (*put/get*)	simple
Data locality information	obscure	explicit	explicit	available
Scalable performance	limited	very good	very good	varies

Note: 1-sided MPI-2 is one of several and not highest performing instance of 1-sided message passing

Holy Grail for Programming Models

Generality/Abstraction/Scalability

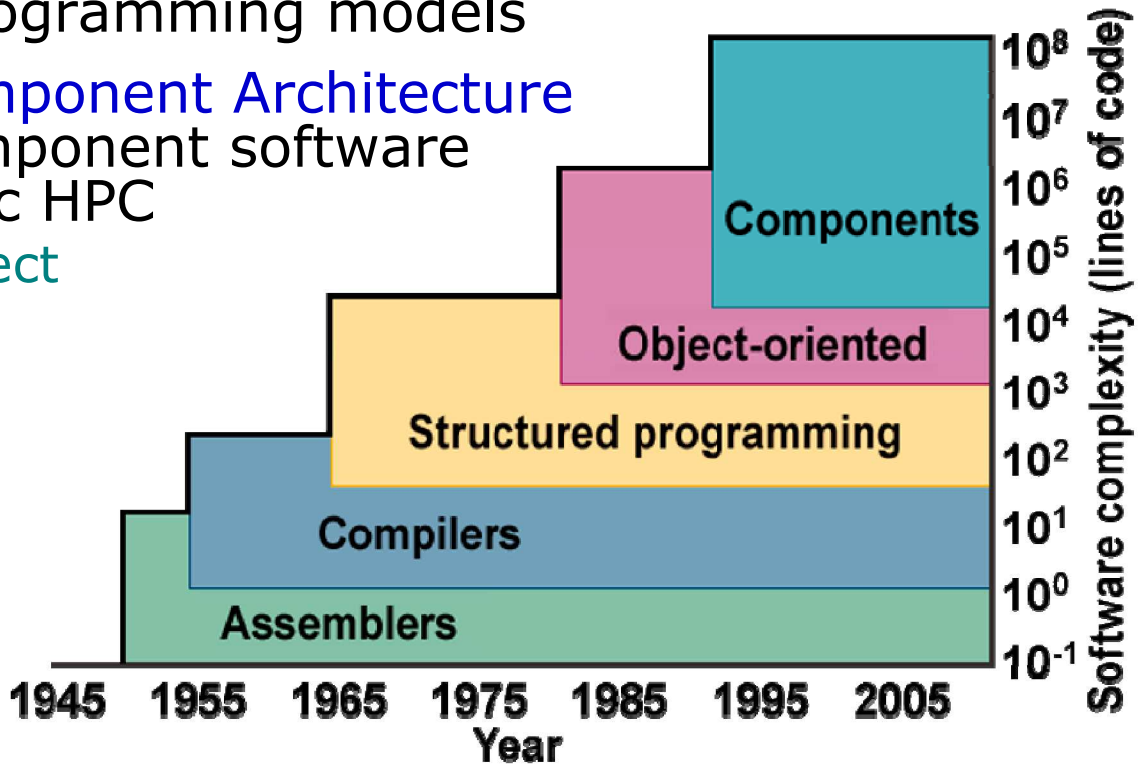


Towards Holy Grail

- Research, software product, users
  - Basic research beyond DARPA HPCS effort needed
    - The current efforts vendor centric, too short term
  - Recognize differences between Model and Implementation
- To deliver production quality adequate level of funding is required to deliver optimal performance
  - Intel example: 4,000 people working on compilers and related s/w tools
  - Compare against DARPA HPCS language teams with handful people involved

From Programming Models to Component Software

- Complexity of scientific software increases with simulation fidelity, multi-physics coupling, computer power
- Applications increasingly often adopt modular design with modules (components) written using different languages and programming models
- The Common Component Architecture (CCA) brings component software approach scientific HPC
  - DoE Scidac project

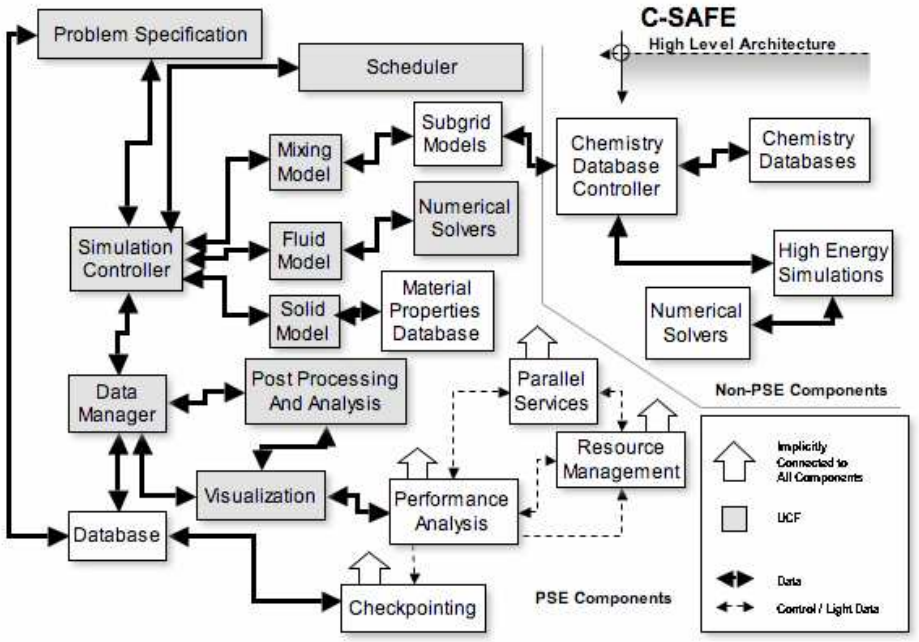


CCA
Common Component Architecture

Components Software

Components represent reusable chunk of software

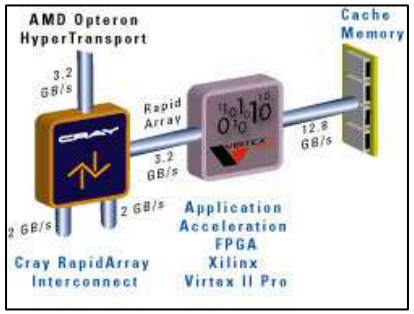
- Inside: private implementation development details
- Outside: publicly accessible functionality (i.e. interfaces)
- Enforces discipline/ responsibility boundaries
- "Firewalls" code, enables easy replacement
- Helps to deal with heterogenous h/w including acclerators

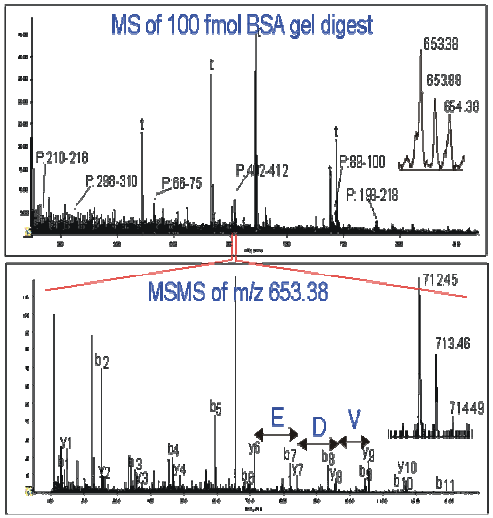


Component-Based Application

CCA on Heterogenous Systems

- With CCA we can build reusable h/w-accelerated components that encapsulate common scientific algorithms
- HPC computers with hardware accelerators connected via a high-speed interconnect:
  - Field Programmable Gate Array (FPGA), Clearspeed, GCPU, Cell
  - Example: proposed LANL IBM Thunder system
- Application accelerator services to accelerate application performance on hybrid or heterogeneous architectures
  - Working with proteomics application Polygraph that uses FPGA computes a "spectral fingerprints"
- CCA Event service used for linking components on general purpose CPU and FPGA

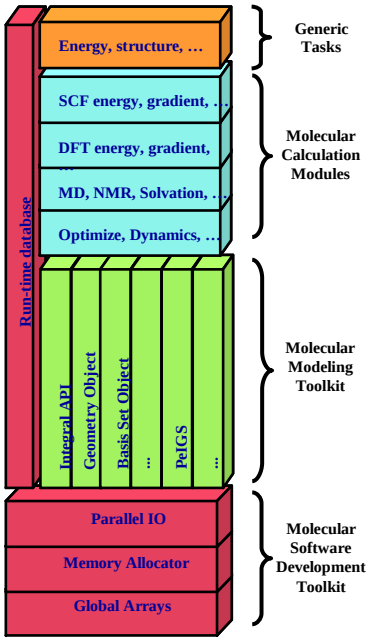




Application Frameworks: NWChem Example

NWCHEM
HIGH-PERFORMANCE COMPUTATIONAL CHEMISTRY SOFTWARE

- A major scalable chemistry package developed 10y ago
- 1.5million lines of code in NWChem
- Development from scratch of a large software suite
- Primary focus on MPP platforms required addressing
  - Scalability
    - At that time most computational chemistry packages used replicated data structures and algorithms with limited scaling
  - Productivity
    - Shortage of highly skilled system experts
    - Developers were chemistry scientists at lab and collaborating universities, postdocs, students not skilled in HPC
  - CS and domain scientists working together
- Decision was to pursue approach much different from ASCI software development model (i.e., based on MPI/OpenMP standards and laborious effort)
  - Develop parallel framework and abstractions in support of domain algorithms: MA, GA, DRA, RTDB
  - Ease of use and high productivity



Final Thoughts

- Changes of hardware and applications ask for advancement of programming models
  - Fault tolerance, scalability, system balance, …
- Research opportunities
  - Should be a part of future US, EU investments in software discussed recently
- Application Frameworks combined with Component Software technology can deliver
  - High-level abstractions
  - Hide implementation and h/w complexities
- Concerted multidisciplinary team effort is needed to achieve science goals for petascale systems

Acknowledgments

- US DoE Office of Advanced Scientific Computing Research (ASCR) Funding
  - Programming Models for Scalable Parallel Computing
  - Scidac CCA project
  - FASTOS Scalable Fault Tolerance project