Hybrid MultiCore: Projects at NASA Ames Research Center

Piyush Mehrotra
NASA Advanced Supercomputing (NAS) Division
NASA Ames Research Center
Piyush.Mehrotra@nasa.gov

HMC Annual Workshop
January 20, 2010

High End Computing Capability @
NASA Advanced Supercomputing (NAS) Division

Supercomputing Systems

- Pleiades: 56,320-core SGI Altix ICE (Xeon)
    - 11,776 quad-core Intel Harpertown (47104 cores)
    - 2,304 quad-core Intel Nehalem (9216 cores)
- Columbia: 13,312-processor SGI Altix (Itanium2)
- RTJones: 4,096-core SGI Altix ICE (Xeon Clovertown)
- Schirra: 640-processor IBM Power5+
- hyperwall2: 1,024-cores, 128-node GPU cluster
- Multiple secure front ends, metadata servers, object storage servers

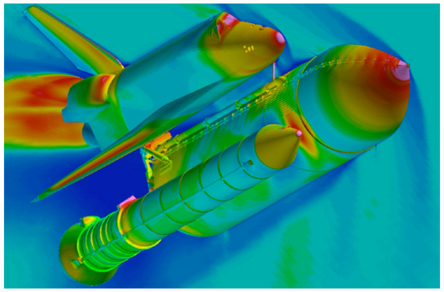
Balanced Environment

- 6 PB disk filesystem; 20 PB tape archive
- Archiving 500TB – 1PB every month
- High-bandwidth WAN to other Centers, external peering
- Large-scale rendering, concurrent visualization

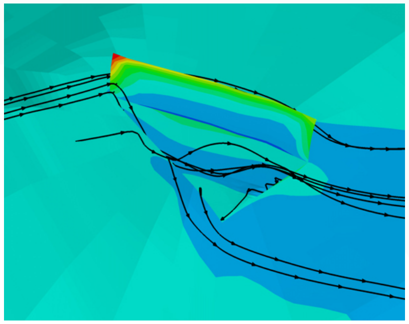
Resources Enable Broad Mission Impact

- Mission Directorates select projects, determine allocations
- More than 450 science & engineering projects
- Approximately 1,200 user accounts
- Typically 400-500 jobs running 24x7

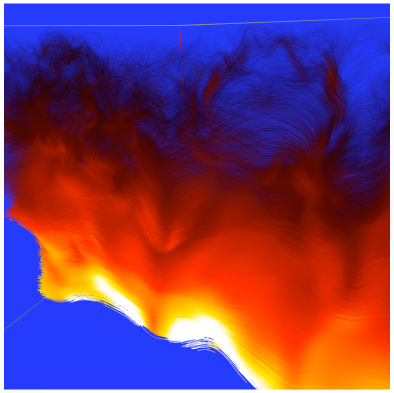
Recent HECC Support for NASA Projects



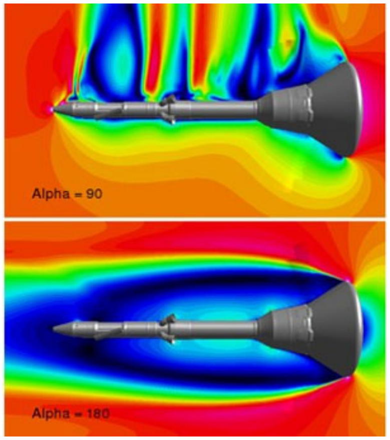
SOMD: Shuttle Aerodynamics



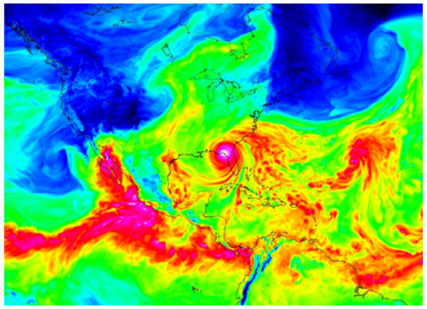
SOMD: Shuttle Damage Analysis



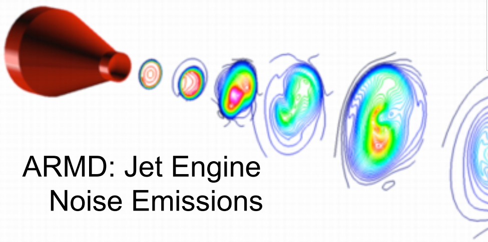
SMD: Solar Surface Convection



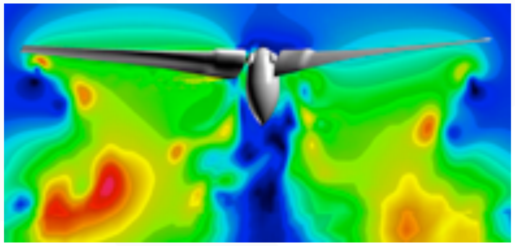
ESMD: Orion Launch Abort



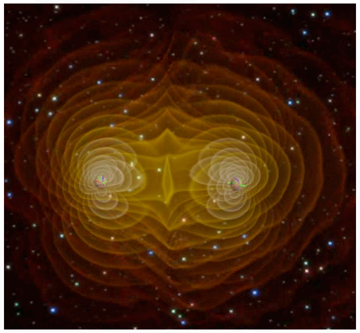
SMD: Hurricane Prediction



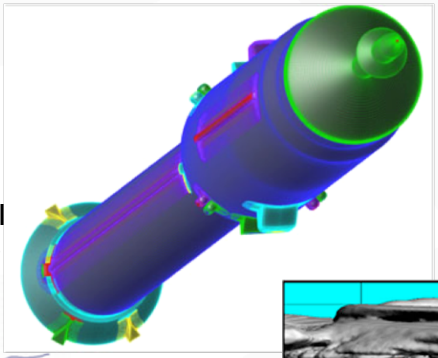
ARMD: Jet Engine
Noise Emissions



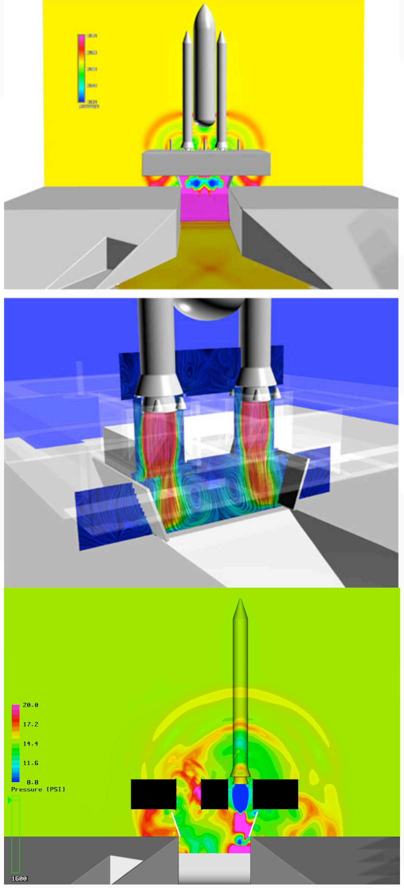
ARMD: V22 Tiltrotor



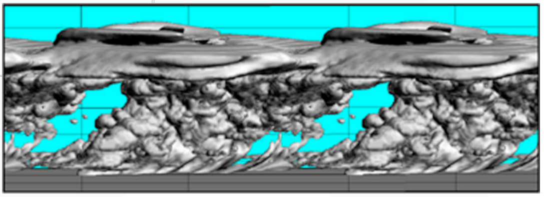
SMD: Spinning Black Holes



ESMD: Ares I
Aerodynamic Database



ESMD: Flame Trench



ARMD: Jet Aircraft Wake Vortices

Code Diversity: Application Landscape

Programming Paradigm	
MPI	most codes
OpenMP	Radhydro (SMD), MAGIC (SMD), IRFS (ARMD)
Serial	NEQAIR (ARMD), Matlab (multiple), LCROSS (ESMD)
MPI+OpenMP	fvGCM (SMD), GISS (SMD), OVERFLOW (ARMD, ESMD, SOMD)
Programming Language	
Fortran77/90/95	most codes / FUN3D (ARMD) / R-WENO(ARMD)
C	Cart3D (ARMD, SOMD)
C++	Cosmos (SMD), NAMD (ESMD), MHDAM (SMD)
Mixed C/Fortran	DAKOTA (ARMD, ESMD), PARK (SMD), RAMS (SMD)
Origin of Code	
Community	most CFD codes, most climate codes, VASP (SMD)
Home-Grown	Radhydro (SMD), MoSST (SMD), ART_MPI (SMD)
ISV	Matlab (multiple), LS_Dyna (NESC), Gaussian (ARMD, ESMD, SMD)
Use of Code	
Time-Sensitive Runs	LAURA (ESMD), DPLR (ESMD), Cart3D (SOMD)
Production	PHANTOM (ESMD, SOMD), LOCI_CHEM (ESMD, SOMD), USM3D (ESMD), NCC (ARMD)
Research	Dynamo (SMD), Hahndol (SMD)
Performance Characteristics	
Embarrassingly Parallel	fms_Ensemble (SMD)
Communication Bound	Cart3D - in MG mode (ARMD, ESMD), TASS (ARMD)
I/O Bound	ECCO (SMD), fvGCM (SMD)
CPU Bound	NAMD (ESMD), LCROSS (ESMD)
Memory Bandwidth Intensive	OVERFLOW (ARMD, ESMD, SOMD), Cart3D (ARMD, ESMD)

Number of distinct applications run on NAS HEC resources: > 150

Performance of CFD code Overflow on a GPU Workstation

Dennis Jespersen: dennis.c.jespersen@nasa.gov

- Code uses structured grids - overset zones around subdomains
- Adaptations for GPU implementation
  - Substituted Jacobi for SSOR
  - Replaced 64 bit with 32 bit arithmetic where possible
  - Changes had little effect on convergence rate or accuracy for problems tested
- Approach
  - Hand-translate one subroutine for GPU (Fortran to CUDA)
  - Compute matrices on CPU, transfer to GPU
  - Multiple ways to map grid points to threads and utilize the memory hierarchy
- Two datasets:
  - Turbulent 3D flat plate case: 121x41x81 grid
  - Turbulent 3D duct case: 166x31x49 grid
- Two execution environments:
  - One 2.1 GHz quad-core AMD Opteron 2352 processor 1.35 GHz NVIDIA GeForce 8800 GTX, 128 cores, 768 MB of global memory
  - Two dual-core 2.8 GHz AMD Opteron 2220 processors 1.30 GHz NVIDIA Tesla C1060, 240 cores, 4 GB of global memory

Performance of Overflow on a GPU

	GTX 8800		Tesla	
Algorithm	Plate	Duct	Plate	Duct
SSOR 64 bit CPU	3.51	2.14	3.83	2.33
Jacobi 32 bit GPU	1.43	0.91	1.35	0.76
GPU/CPU	0.41	0.43	0.35	0.49

Time for LHS in sec/step (lower is better)

- Same work: SSOR 10 forward + 10 reverse sweeps; Jacobi 20 sweeps
- Matrix elements computed on front end – transferred to GPU
- GPU time: 2.5–3 times faster than original 64 bit SSOR

	GTX 8800		Tesla	
	Plate	Duct	Plate	Duct
GPU Total	0.904	0.576	0.314	0.193
GPU Kernel	0.784	0.499	0.142	0.082
Overhead	0.124	0.077	0.172	0.111

GPU Time in sec/step (lower is better)

- Overhead including data transfer time ranges from 15% to 135%

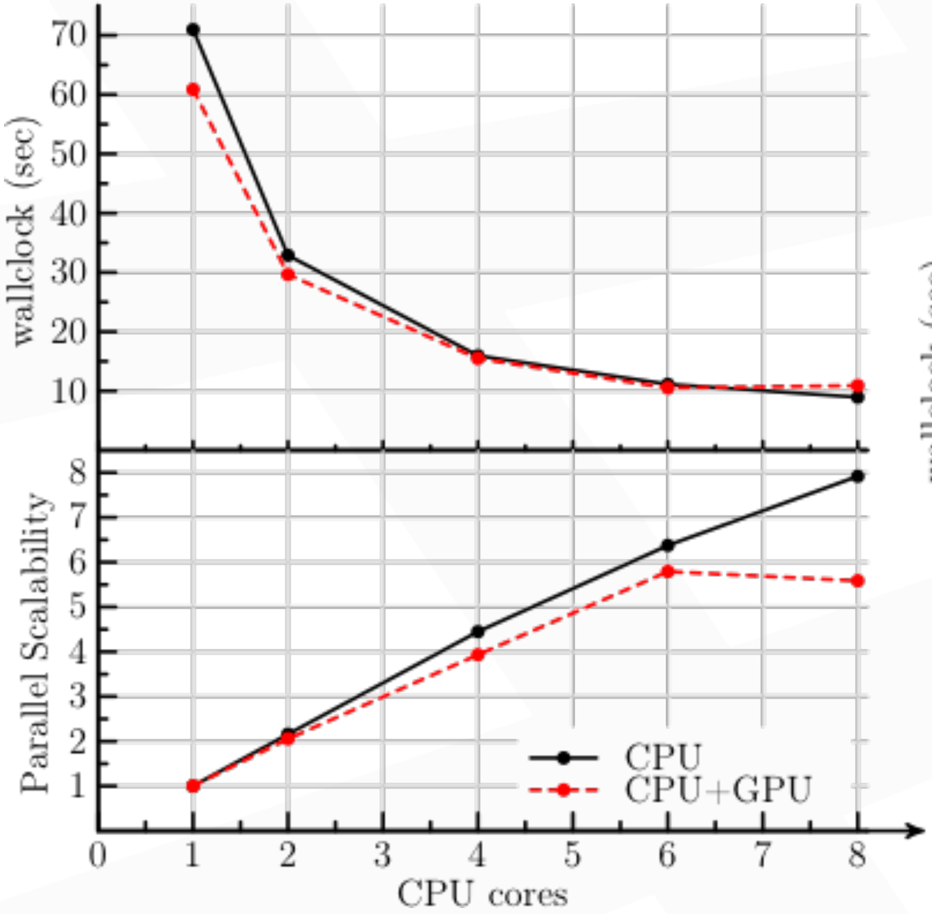
Results published in ParCFD 2009

Performance of a DNS code on a GPU Cluster

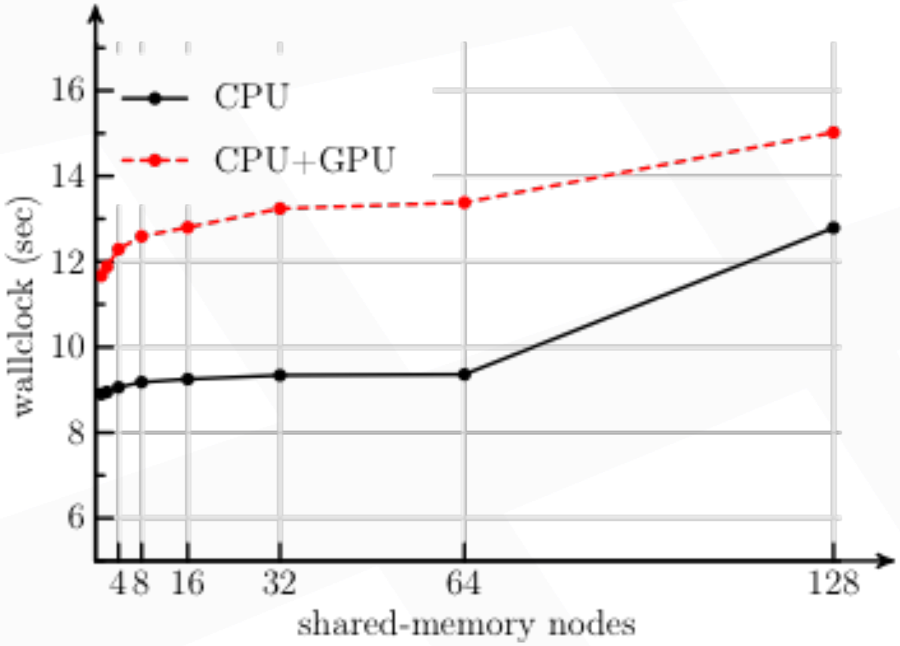
Scott Murman: scott.murman@nasa.gov

- Specialized DNS turbulence solver using spectral and high-order central derivatives
  - Co-process (expensive) spectral operators on GPU (cuFFT) concurrently w/ central operators on multiple CPU cores
  - Hybrid programming model - shared memory for GPU access using OpenMP w/ MPI distributed layer
- Experiments on *hyperwall*
  - 128 nodes, 2x4 AMD Opteron 2.2GHz
  - Each node w/ NVIDIA GeForce 8800 GTX
  - Fat-tree IB network

DNS Solver - Preliminary Results



- Strong Scaling within a node
- Spectral operator on GPU – central differencing on CPU
- Best performance @ 4 cores



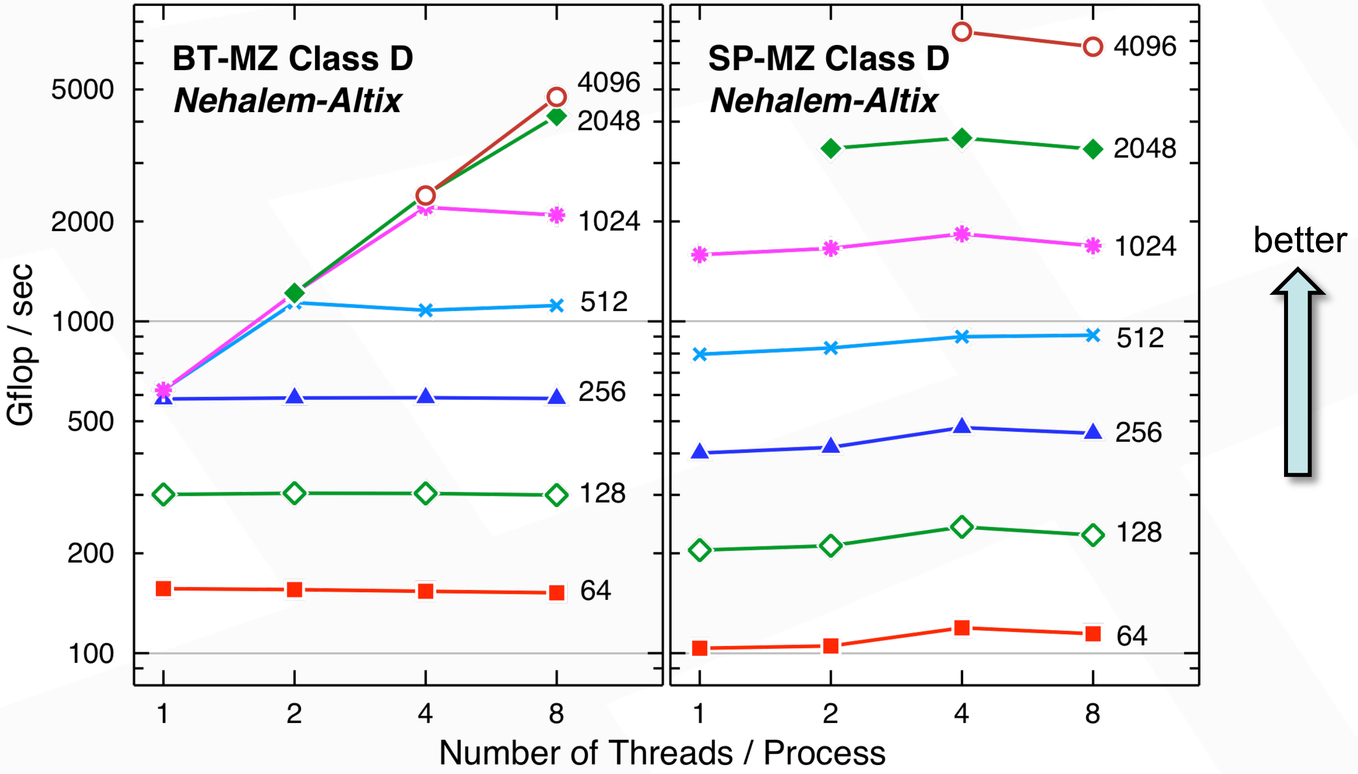
- Weak Scaling across nodes (8 cores per node)
  - Total problem size of 2.2B DoF on 128 nodes
- Scalability using GPUs tracks CPU only

GPU Effort conclusions

- Code acceleration is possible for CFD codes – may require algorithmic rethinking
- 32-bit arithmetic need not be a stumbling block
- Issues:
  - Libraries may still be immature
    - CUDA FFT library - performance bottleneck for DNS solver
  - Requires efficient management of memory hierarchy
    - Need support for “automatic” mapping
  - Portability/maintainability is still a major issue

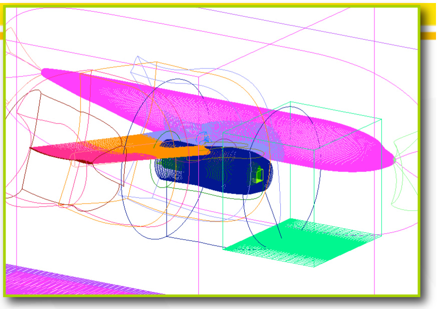
Hybrid Programming on MultiCores: Multi-Zone NAS Parallel Benchmarks

Henry Jin: Henry.Jin@nasa.gov

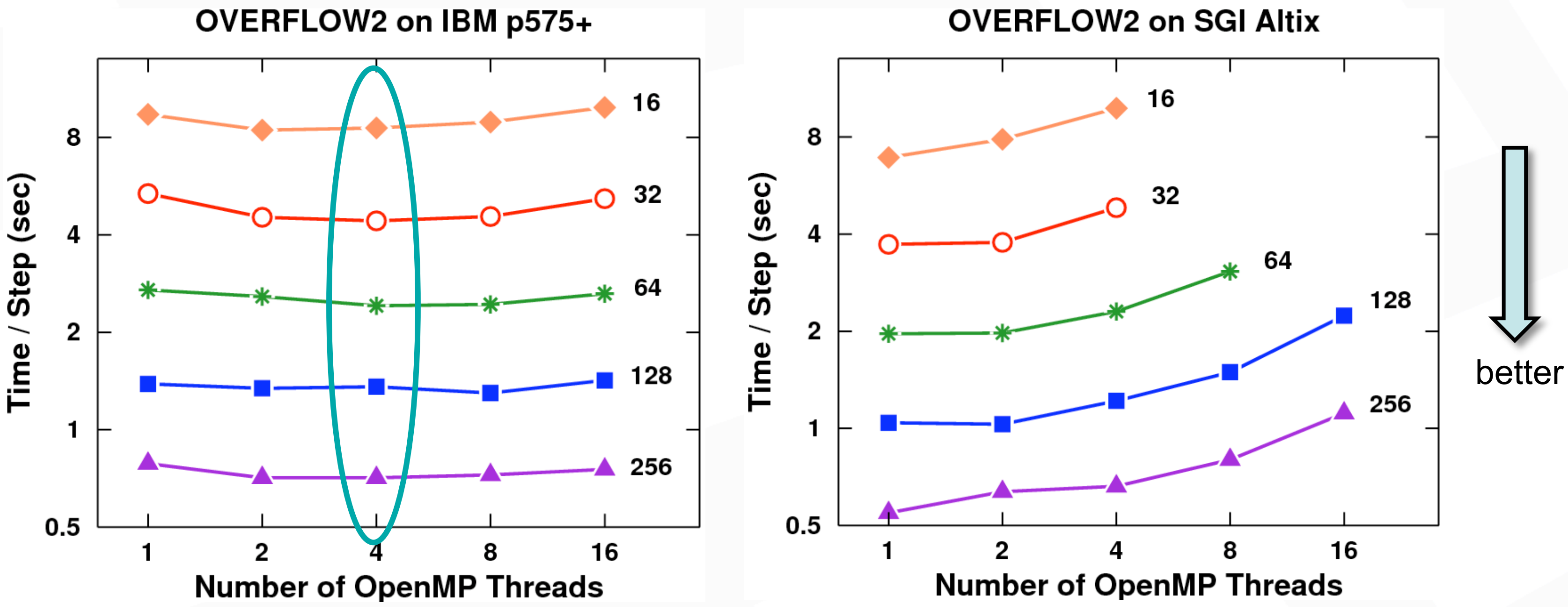


- BT-MZ: zones with uneven sizes, SP-MZ: zones with same size
- Hybrid parallelization: MPI exploits coarse grained parallelism among zones, while OpenMP applies to loop level parallelism within each zone
- MPI is limited by the number of zones and load imbalance, while OpenMP improves load balance (at large CPU counts) and cache utilization (in SP-MZ)

Hybrid Programming on MultiCores: OVERFLOW2 - DLRF6 Case



Dennis Jespersen: dennis.c.jespersen@nasa.gov



36 M grid points, 23 zones, DLRF6 benchmark configuration

- MPI+OpenMP version: numerically explicit scheme + implicit scheme. Implicit scheme has faster convergence rate and reduces the total number of grid points
- Hybrid version outperformed pure MPI version on the IBM p575+
- The benefit of OpenMP in the hybrid mode on IBM p575+ does not show on the SGI Altix, although the pure MPI version performed better on the Altix

Locality-Aware Computation in OpenMP

Henry Jin: Henry.Jin@nasa.gov

- Joint NSF project with B. Chapman and L. Huang (University of Houston)
- Current OpenMP assumes a flat memory space, but in reality it is often not
- Introduce locality aware into OpenMP
  - Define logical *locations* for OpenMP tasks
    - Derived from the HPCS languages
  - Distribute shared data among locations
  - Tie OpenMP tasks with locations through the use of clause “**`onloc`**” to **`omp parallel`** or **`omp task`**
- Prototype implementation using the research compiler (OpenUH)

Differential Performance Analysis for Multicore systems

Hood, Jin, Mehrotra, Biswas, Chang, Djohmehri, Gavali, Jespersen, Taylor

- Contention for resources on multi-core nodes
- Performance of Overflow across architectures *(dataset DLFR6)*:

# of Processes	**C24** **(Itanium2)**	**Pleiades (Harpertown)**		**Pleiades (Nehalem)**	
		8 GB/node		*24 GB/node*	
		8ppn	*4ppn*	*8ppn*	*4ppn*
16	6.87	16.24	7.29	6.81	4.84
32	3.74	6.96	3.40	3.24	2.41
64	1.93	3.09	1.75	1.58	1.28
128	1.01	1.49	0.91	0.80	0.70
256	0.51	0.74	0.47	0.42	0.38

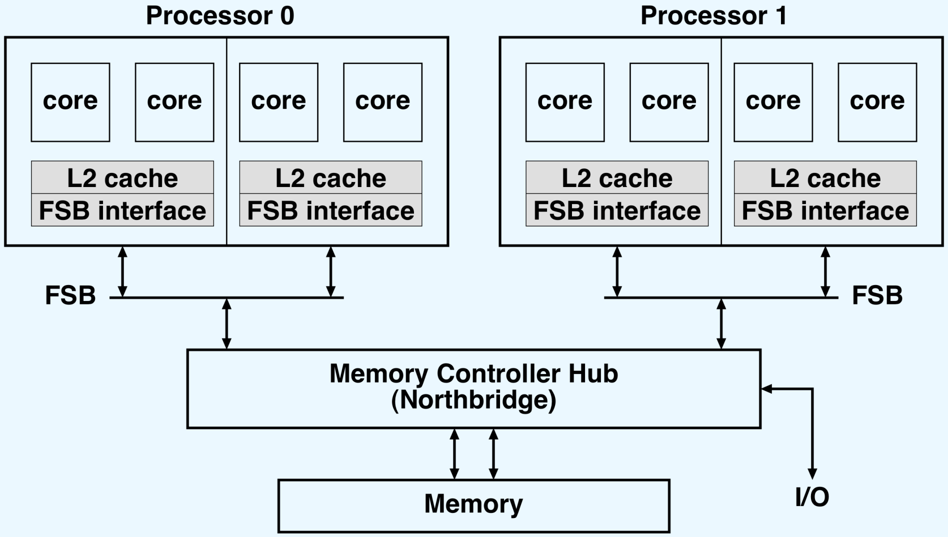
- Superlinear behavior?: cache, memory bandwidth effect
- Also note: some 4ppn on Pleiades > twice as fast as 8ppn
- *Differential performance analysis*:
    - A methodology to isolate performance impact of resource sharing
    - Allow users to identify effect of resource contention without modification or instrumentation of code

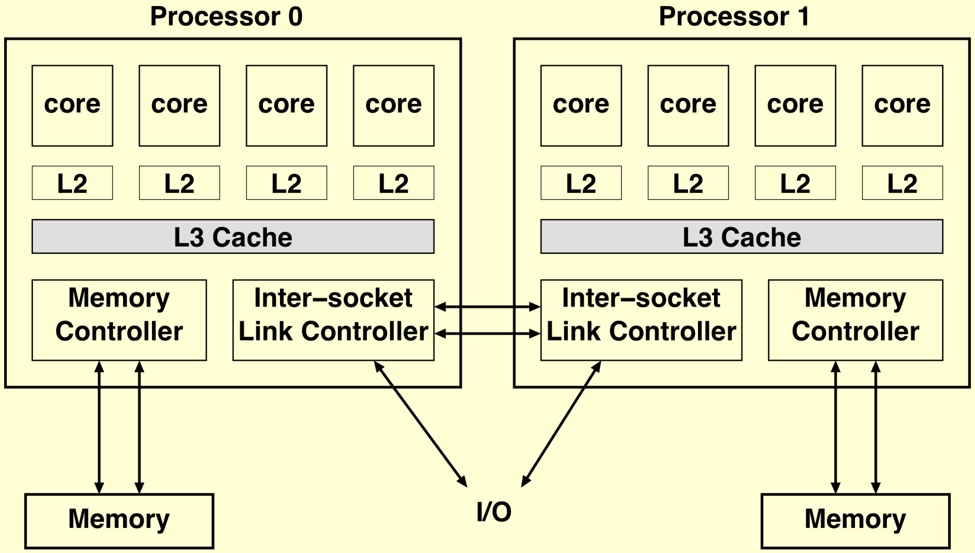
Sharing in Multicore Node Architectures

UMA Based

Harpertown / Clovertown

- L2
- FSB
- Memory Controller





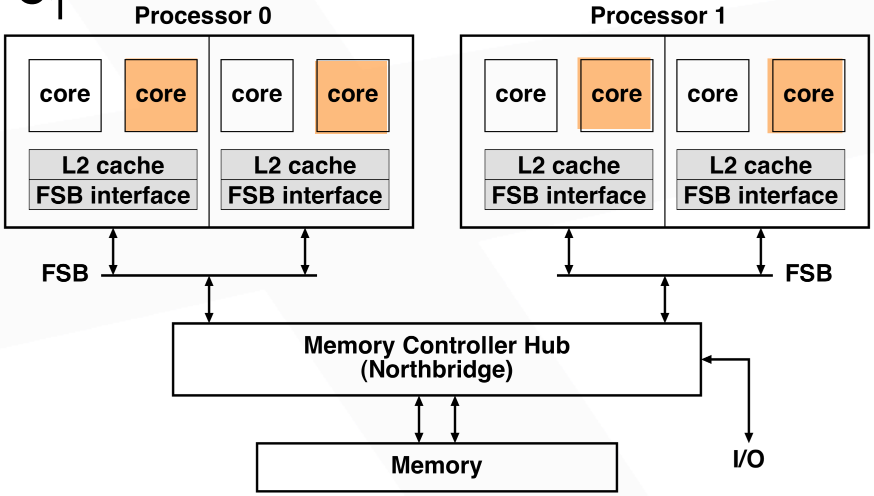
NUMA Based

Opteron / Nehalem

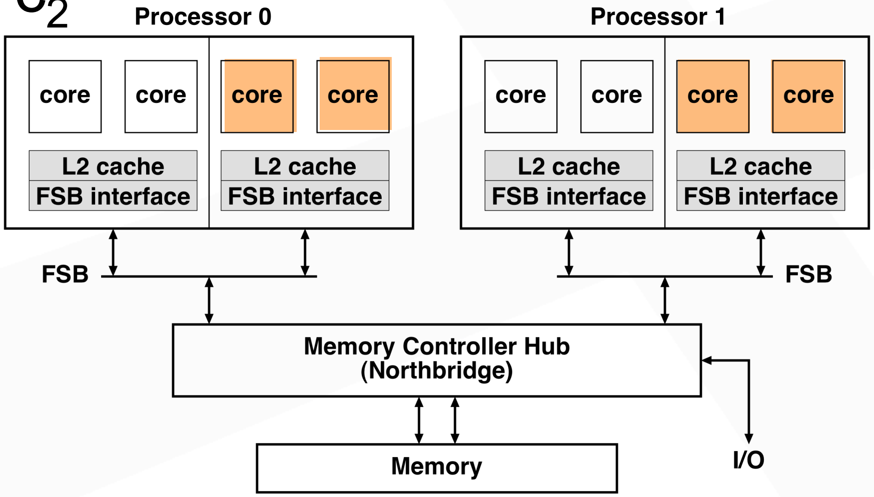
- L3
- Memory Controller
- HT3 / QPI

Isolating Resource Contention

C_1



C_2



- Evaluate performance of code on configurations with specified mappings of processes to cores
- Compare two configurations (e.g., C1 & C2) of MPI processes assigned to cores
  - Both use 4 cores per node
  - Communication patterns the same
  - Equal loads on: FSB & MC
  - Difference is in sharing of L2
- Compare timings of runs using these two configurations
  - Performance penalty to identify impact of sharing L2

$$P_{C1\rightarrow C2} = (t_{C2} - t_{C1})/t_{C1}$$

- Other pattern pairs can isolate FSB, memory controller

Impact of Resource Sharing on Performance

Penalty for Sharing Resource	Cart3D	OVERFLOW	MITgcmuv
Harpertown			
○ L2 cache	2 – 4%	40%	24%
○ Front-side bus	22 – 41%	24 – 54%	50-71%
○ Memory controller	0 – 5%	1 – 3%	5-6%
Nehalem			
○ L3 cache + memory controller	2 – 5%	8 – 34%	27-72%
○ QPI	2 – 23%	0 – 4%	1-4%

- Each penalty calculated using 2 or 4 pairs of related configurations giving rise to ranges
- Cart3D optimized for cache, however is a scarce resource for Overflow and MITgcmuv
- FSB is an issue with each of the codes whereas the memory controller (shared by both sockets) is not
- On the Nehalem, the L3 cache and MC is a bottleneck for MITgcmuv and also for Overflow whereas the QPI is not

To be published in IPDPS 2010

Future Plans

- Continue investigating optimal mapping of CFD and other codes on GPUs and other accelrators
- Evaluate “many”” core systems from Intel, AMD, IBM including SGI’s UltraViolet
- Evaluate mixed programming models
- Locality aware extensions of OpenMP
- Extend differential performance analysis to isolate communication effects

piyush.mehrotra@nasa.gov