The MareIncognito Project

Jesus Labarta
Director Computer Sciences Research Dept.
BSC

Objective

- Design a 10+ Petaflops Supercomputer for 2010-11
- - IBM Cooperation
- Spanish position with PRACE

Mare Incognito

"Homogeneous" Supercomputer based on the Cell processor

We believe it is possible to build

"cheap", "efficient", "not application specific" and

"widely applicable" machines

We know it is risky

We have "a vision" of relevant technologies to develop

Many of them are not Cell specific and will be evaluated for other architectures

The Opportunity

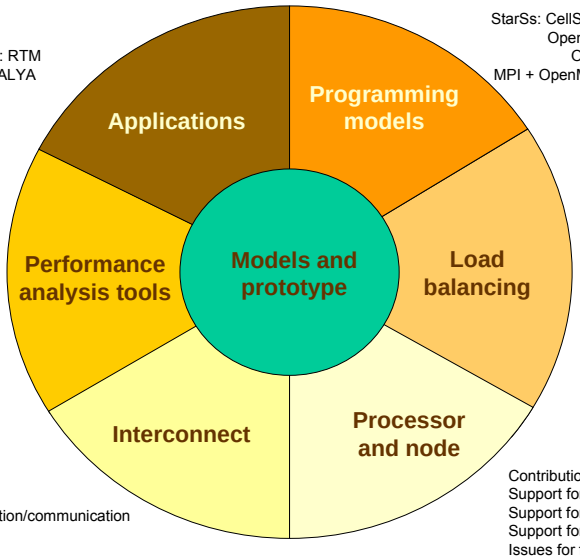
To integrate all research lines within BSC and to increase our cooperation with IBM

Influence the design and use of supercomputers in the future

MareIncognito: Project structure

4 relevant apps:
Materials: SIESTA
Geophisics imaging: RTM
Comp. Mechanics: ALYA
Plasma: EUTERPE
General kernels

StarSs: CellSs, SMPSs
OpenMP@Cell
OpenMP++
MPI + OpenMP/StarSs



Automatic analysis
Coarse/fine grain prediction
Sampling
Clustering
Integration with Peekperf

Coordinated scheduling:
Run time,
Process,
Job
Power efficiency

Contention
Collectives
Overlap computation/communication

Contribution to new Cell design
Support for programming model
Support for load balancing
Support for performance tools
Issues for future processors

Vision, work and findings

- General Concerns:
  - Heterogeneous / hierarchical / dynamic trend
  - Memory
  - Variance
  - Are we overdimensioning our systems?
  - Globalization
    - Holistic design, Butterfly effect
- Workpackages
  - Programming models
  - Node design
  - Load balance
  - Interconnects

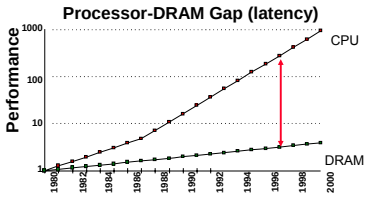
Heterogeneous, hierarchical and dynamic environment

- Foreseeable plethora of architectures
  - Thick nodes
  - Driven by what can be done
- Heterogeneous
  - Functionality
  - Performance
    - On purpose
    - Result of manufacturing process
    - Result of infrastructure construction process (~no cathedrals in pure architectonic style)
- Hierarchical
  - Can not provide flat uniform view (latency and bandwidth)
  - Hierarchical domains support different granularities
- Dynamic
  - Application characteristics
  - Workload
  - Resource allocation practices

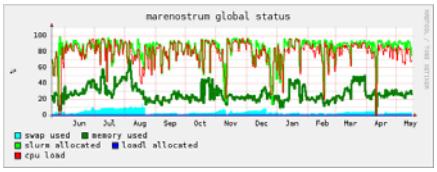
Memory: more than a wall

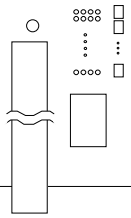
- Performance:
  - Latency
  - bandwidth
- Cost
- Power
- Capacity
  - Real usage < 40% ?
  - Accelerator model → 2x ?

- Main component/nightmare of programming model



D.A. Patterson " New directions in Computer Architecture" Berkeley, June 1998



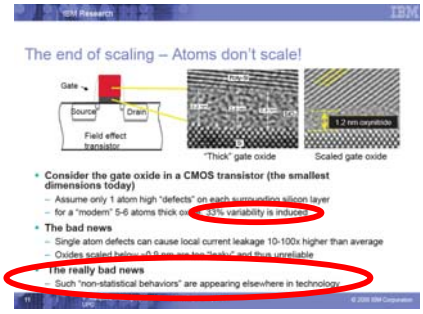


Fighting variance: a lost battle

- We are going to experience huge variability
  - In resources (availability, performance) and usage needs
  - In space and time

- Better learn how to tolerate it

- How to face it
  - Adaptive/Dynamic resource management
    - Load balance
  - Asynchronism



Stolen from: V. Salapura "Scaling up next Generation Supercomputers". UPC

Are our designs an overkill?

- Are we using resources efficiently?
- Resources:
  - Processors
  - Memory
  - interconnect
  - **Energy**

"To kill flies with guns"

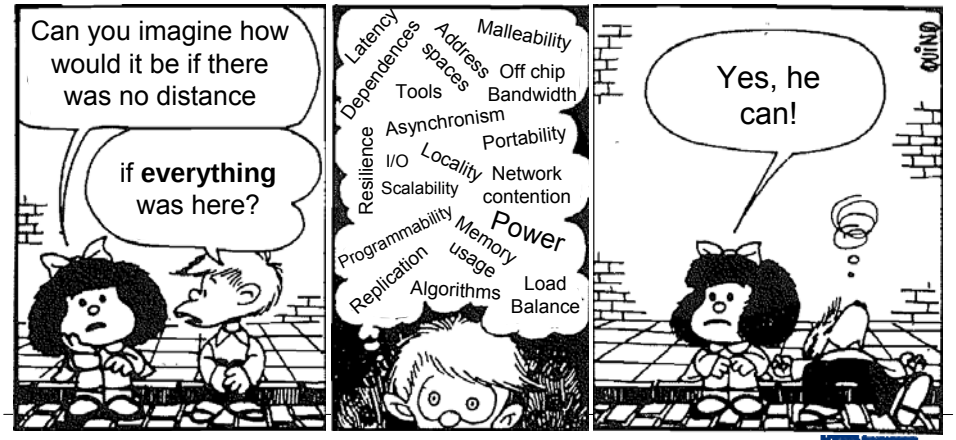
The butterfly effect

- Sensitivity to initial conditions
- Huge impacts of small causes
- High non linearities with accumulative effects

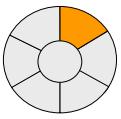
"Does the flap of a butterfly's wings in Brazil set off a tornado in Texas?"

Globalization: Holistic approach

- Can we develop a unified theory/model? Nicely integrate all levels and experiences?
- How do we ensure coordination/cooperation between levels at run time?



Programming model



Back to Babel?

Book of Genesis

"Now the whole earth had one language and the same words" …

…"Come, let us make bricks, and burn them thoroughly. "…

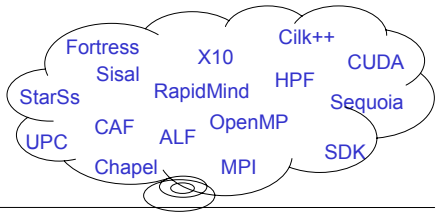
…"Come, let us build ourselves a city, and a tower with its top in the heavens, and let us make a name for ourselves"…

And the LORD said, "Look, they are one people, and they have all one language; and this is only the beginning of what they will do; nothing that they propose to do will now be impossible for them. Come, let us go down, and confuse their language there, so that they will not understand one another's speech."

The computer age

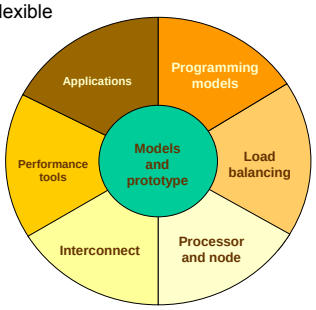
Fortran & MPI

++

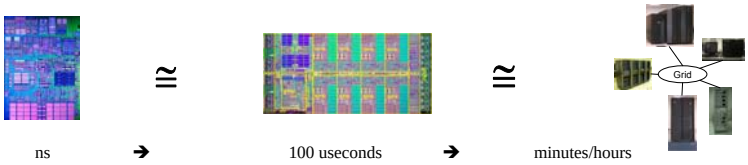


Programming model for MareIncognito?

- How much pressure can we put on our (BSC) program developers?
- Is effort worthwhile? "long term"?
  - Only once porting effort
  - Can not keep spending 6 months and increasing by 15% the number of lines for each new target machine
- Need smooth transition path
  - Can not fire our programmers and they are often not very flexible
- can afford work on "beneficial" transformations
  - Blocking
  - Better understanding of inputs and outputs
  - Understanding potential asynchrony
- Evolution towards mixed mode
  - MPI+OpenMP, **MPI + StarSs**, MPI+StarSs+OpenCL, …
  - Matches hierarchy in architectures, algorithmic,
  - Other potential benefits: load balance.



A perspective on architectures and programming models



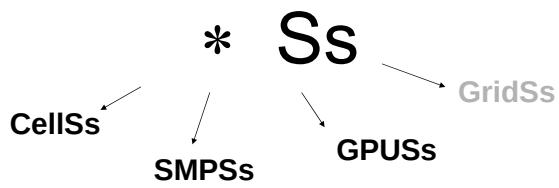
ns ➔ 100 useconds ➔ minutes/hours

Mapping of concepts:

Instructions	➔ Block operations	➔ Full binary
Functional units	➔ SPUs	➔ machines
Fetch &decode unit	➔ PPE	➔ home machine
Registers (name space)	➔ Main memory	➔ Files
Registers (storage)	➔ SPU memory	➔ Files

Granularity
Stay sequential
Just look at things from a bit further away
Architects do know how to run parallel

StarSs



- Programability
  - Standard sequential look and feel (C, Fortran)
  - Incremental parallelization/restructure
  - Abstract/separate algorithmic issues from resources
  - Methodology/practices
    - Block algorithms: modularity
    - "No" side effects: local addressing
    - Promote visibility of "Main" data
    - Explicit synchronization variables
- Portability
  - Runtime for each type of target platform.
    - Matches computations to resources
    - Achieves "decent" performance
  - Even to sequential platform
  - Single source for maintained version of a application
- Performance
  - Runtime intelligence

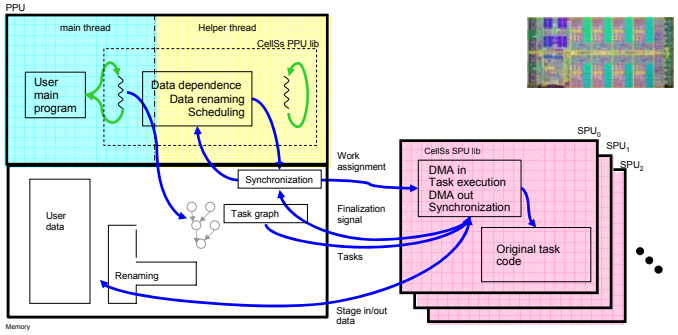
Cell superscalar (CellSs)

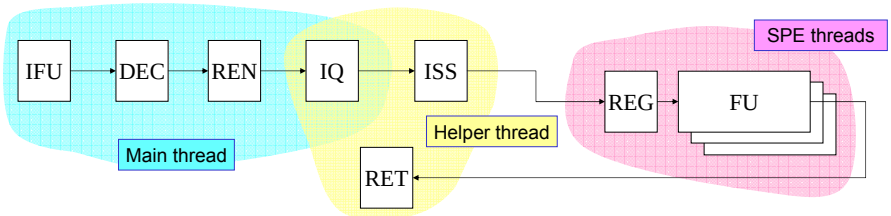
- Directives to define tasks in sequential block algorithm
- Automatic parallelism exploitation at run time

```
int main(){
…
for (i=0; i < N; i++)
  for (j=0; j < N; j++)
    for (k=0; k < N; k++)
      block_addmultiply( C[i][j], A[i][k], B[k][j]);
...
```

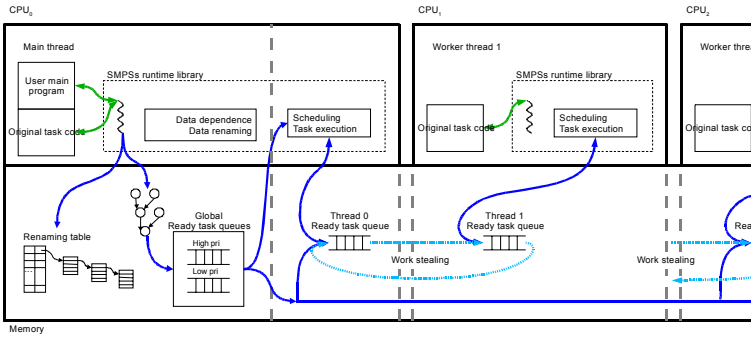
```
#pragma css task input(A, B) inout(C)
static void block_addmultiply( float C[BS][BS], float A[BS][BS],  float B[BS][BS]) {
…
for (i=0; i < BS; i++)
  for (j=0; j < BS; j++)
    for (k=0; k < BS; k++)
      C[i][j] += A[i][k] * B[k][j];
}
```

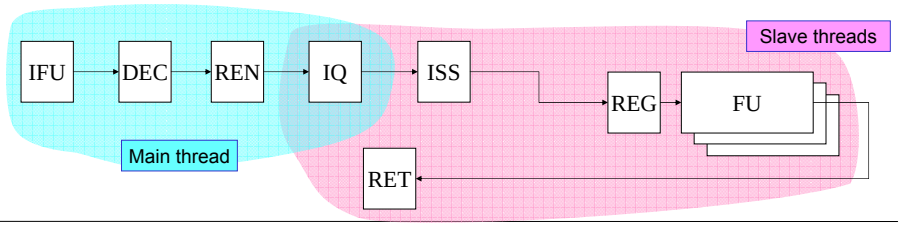

CellSs execution model





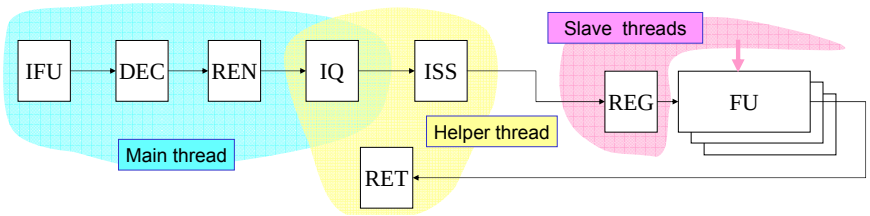
SMPSs execution model





GPUSs

- Architecture implications
  - Large local store O(GB) → large task granularity ← Good
  - Data transfers: Slow, non overlapped ← Bad
- Cache management
  - Write-through
  - Write-back
- Run time implementation
  - Powerful main processor and multiple cores
  - Dumb accelerator (not able to perform data transfers, implement software cache,…)



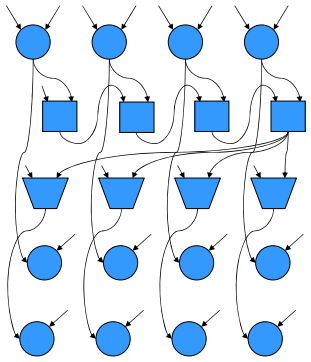
E. Ayguade, et al, "An Extension of the StarSs Programming Model for Platforms with Multiple GPUs"

StarSs

```
#pragma css task input(A, B) output(C)
void vadd3 (float A[BS], float B[BS],
            float C[BS]);
#pragma css task input(sum, A) output(B)
void scale_add (float sum, float A[BS],
                float B[BS]);
#pragma css task input(A) inout(sum)
void accum (float A[BS], float *sum);
```

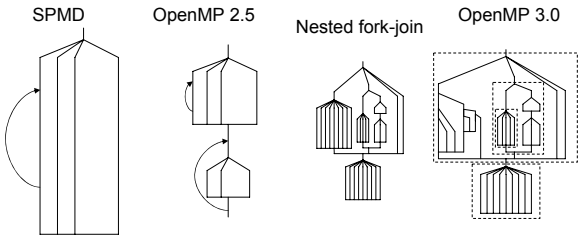
```
for (i=0; i<N; i+=BS)             // C=A+B
   vadd3 ( &A[i], &B[i], &C[i]);
...
for (i=0; i<N; i+=BS)            // sum(C[i])
   accum (&C[i], &sum);
...
for (i=0; i<N; i+=BS)            // B=sum*A
   scale_add (sum, &E[i], &B[i]);
...
for (i=0; i<N; i+=BS)            // A=C+D
   vadd3 (&C[i], &D[i], &A[i]);
...
for (i=0; i<N; i+=BS)            // E=C+F
   vadd3 (&C[i], &F[i], &E[i]);
```

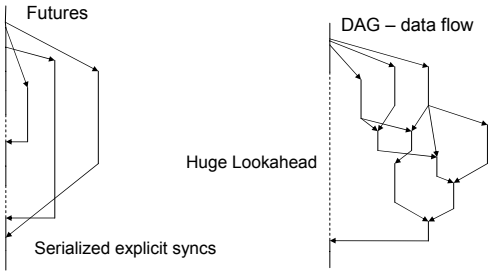
Decouple
how we write
form
how it is executed



Work generation and synchronization

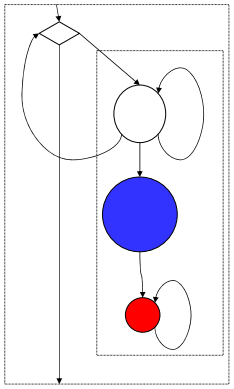
- Need
  - Flexibility
  - Asynchronism





Asynchronism: I/O

- Sequential file processing
  - Statistics
  - Filters



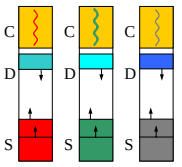
```
#pragma css task inout(fd)
                output(buffer, R_recs, end_trace) highpriority
void Read (FILE *fd, char buffer[500][4096], int *R_recs,
                int *end_trace);

#pragma css task input(buffer_in, R_recs, control)
                output(buffer_out, W_recs)
void Process (char buffer_in[500][4096], int R_recs,
              proc_ctrl control,
              char buffer_out[500][4096], int *W_recs);

#pragma css task input(buffer, W_recs)
                inout(fd, total_records)
void Write (FILE *fd, char buffer[500][4096], int W_recs,
           unsigned long long *total_records);

int main()
{  …
 while(!end_trace) {
   Read (infile, buffer_in, &RR, &end_trace);
   Process (buffer_in, RR, control, &RW, buffer_out);
   Write (outfile, buffer_out, RW, &total_records);
   #pragma css wait on (&end_trace)
 }
}
```


Address spaces: a real problem

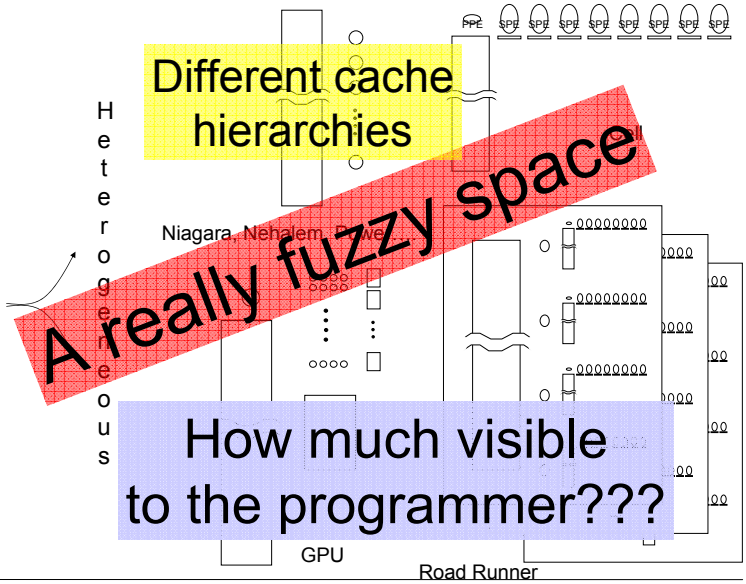


Homogeneous in structure

Heterogeneous

Different cache hierarchies

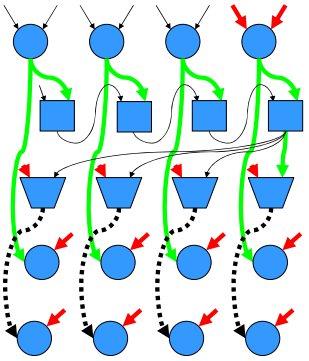
A really fuzzy space



How much visible to the programmer???

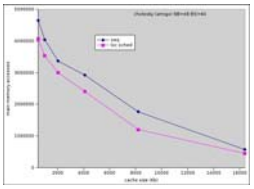
StarSs: potential of data access information

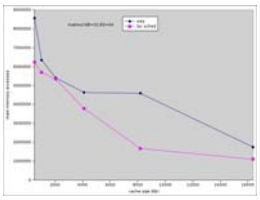
- **Flat global address space seen by programmer**
- Flexibility to dynamically traverse dataflow graph "optimizing"
  - Concurrency. Critical path
  - Memory access
- Opportunities for
  - Prefetch
  - Reuse
  - Eliminate antidependences (rename)
  - Replication management

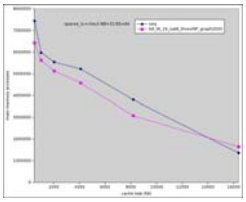


CellSs: fighting the memory wall at runtime

- Object cache: Reuse
  - Reuse SPE Local Store
  - Avoid DMA transfers
- Double buffer: Prefetch
  - Automatic: before executing a task, launch
    - DMA in for arguments for the next
    - DMA out for results of previous
  - Dependent on object sizes – LS pressure
  - Handle alignment issues
- Locality scheduler: alleviate off chip bandwidth bottleneck
  - Depth first
  - Maximum depth
  - Where to resume





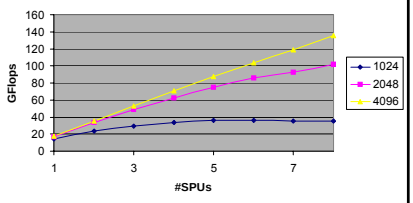


Peter

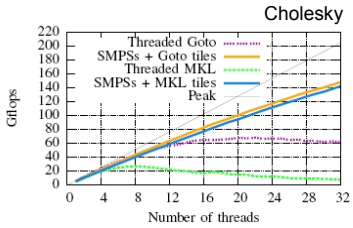
Results

- Fair/good speedups

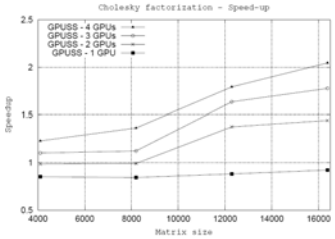
CellSs



SMPSs

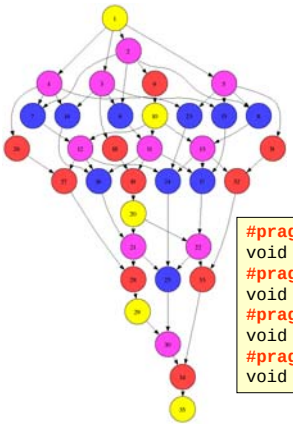


GPUSs



Cholesky: SMPSs

- "Irregular" task graph



```
void Cholesky( float *A ) {
   int i, j, k;
   for (k=0; k<NT; k++) {
      chol_spotrf (A[k*NT+k]) ; // Factorize diag. block

      for (i=k+1; i<NT; i++)
         chol_strsm (A[k*NT+k], A[k*NT+i]); //Triang. solves

      // update trailing submatrix
      for (i=k+1; i<NT; i++) {
         for (j=k+1; j<i; j++)
            chol_sgemm( A[k*NT+i], A[k*NT+j], A[j*NT+i]);
         chol_ssyrk (A[k*NT+i], A[i*NT+i]);
   }
}
```

```
#pragma css task inout (A[TS][TS])
void chol_spotrf (float *A);
#pragma css task input (A[TS][TS]) inout (C[TS][TS])
void chol_ssyrk (float *A, float *C);
#pragma css task input (A[TS][TS], B[TS][TS}) inout (C[TS][TS])
void chol_sgemm (float *A, float *B, float *C);
#pragma css task input (T[TS][TS]) inout (B[TS][TS])
void chol_strsm (float *T, float *B);
```

Cholesky: CellSs, GPUSs

- Stubs for libraries (i.e. Cell, CUBLAS)
- CUDA code

```
void Cholesky( float *A ) {
   int i, j, k;
   for (k=0; k<NT; k++) {
      chol_spotrf (A[k*NT+k]) ; // Factorize diag. block

      for (i=k+1; i<NT; i++)
         chol_strsm (A[k*NT+k], A[k*NT+i]); //Triang. solves

      // update trailing submatrix
      for (i=k+1; i<NT; i++) {
         for (j=k+1; j<i; j++)
            chol_sgemm( A[k*NT+i], A[k*NT+j], A[j*NT+i]);
         chol_ssyrk (A[k*NT+i], A[i*NT+i]);
   }
}
```

```
#pragma css task inout (A[TS][TS])
void chol_spotrf (float *A);
#pragma css task input (A[TS][TS]) inout (C[TS][TS])
void chol_ssyrk (float *A, float *C) {
#pragma css task input (A[TS][TS], B[TS][TS]) inout (C[TS][TS])
void chol_sgemm (float *A, float *B, float *C) {
```

```
#pragma css task input (T[TS][TS]) inout (B[TS][TS])
void chol_strsm (float *T, float *B) {
   float sone = 1.0;
   int ts = TS;
   strsm ("Right", "Lower", "Transpose", "Non-Unit ",&ts, &ts, &sone, T, &ts, B, &ts);
}
```


StarSs: Heterogeneity

- A really heterogeneous system may have several hosts, and different types of accelerators or specific resources
- Different implementations
  - Default: every task should at least be runable on the host
  - implementation for each specific accelerators (even alternative impls.)

```
#pragma css task inout (A[TS][TS])
void chol_spotrf (float *A);
```

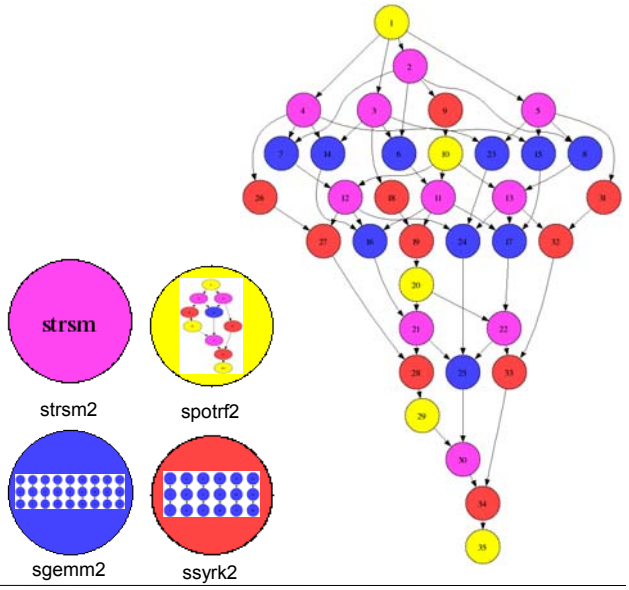
```
#pragma css target device (cell) copyin (T[TS][TS], B[TS][TS]) copyout (B[TS][TS])
#pragma css task input (T[TS][TS]) inout (B[TS][TS])
void chol_strsm (float *T, float *B);

#pragma css target device (cell) copyin (A[TS][TS], C[TS][TS]) \
                                 copyout (C[TS][TS])
#pragma css task input (A[TS][TS]) inout (C[TS][TS])
void chol_ssyrk (float *A, float *C);
```

```
#pragma css target device (cell, cuda) copyin (T[TS][TS], B[TS][TS], C[TS][TS]) \
                                       copyout (B[TS][TS])
#pragma css task input (A[TS][TS], B[TS][TS]) inout (C[TS][TS])
void chol_sgemm (float *A, float *B, float *C);
```


StarSs: hierarchical

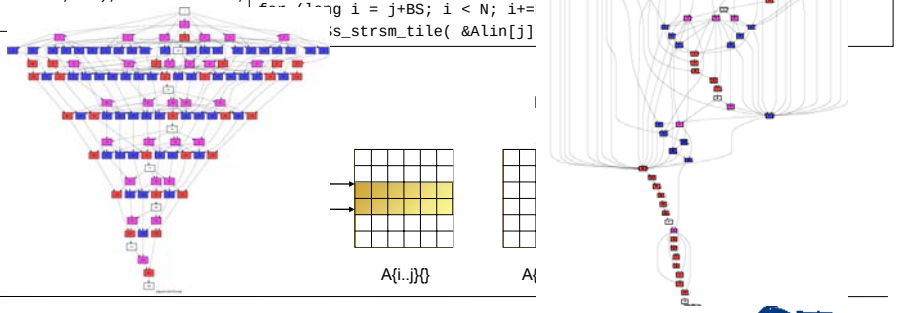
- cholesky



StarSs: array regions

```
#pragma css task input(A[N][N]{0:BS}{0:BS},
B[N][N]{0:BS}{0:BS}, BS, N) inout(C[N][N]{0:BS}{0:BS})
void sgemm_tile(float  *A, float *B, float *C, integer BS,
integer N) {
   unsigned char TR='T', NT='N';
   float DONE=1.0, DMONE=-1.0;
   sgemm_(&NT, &TR,        /* TRA
           &BS, &BS, &BS,    /* M
           &DMONE,           /* A
           A, &N,            /* A,
           B, &N,            /* B,
           &DONE,             /* B
           C, &N);           /* C,
}
```

```
for (long j = 0; j < N; j+=BS
   for (long k= 0; k< j; k+=BS
      for (long i = j+BS; i <
         sgemm_tile( &Alin[k][
for (long i = 0; i < j; i+=BS)
    smpSs_ssyrk_tile( &Alin[i]
smpSs_spotrf_tile( &Alin[j][j]
for (long i = j+BS; i < N; i+=
    smpSs_strsm_tile( &Alin[j]
```



Comparison to OpenMP

OpenMP

Explicit parallelism.

Fork join

Tasks provide some more flexibility

No locality information. Global Addressing

Nesting

StarSs

Implicit parallelism
"atomic" tasks

Explicit data access information
Local addressing

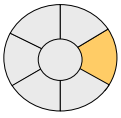
Single work generator

Cross pollination

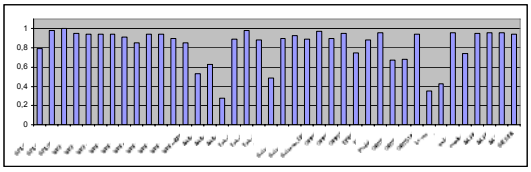
Proposals for standard extension

E. Ayguade, et all, "Extending OpenMP to Survive the Heterogeneous Multi-core Era"

Load balance



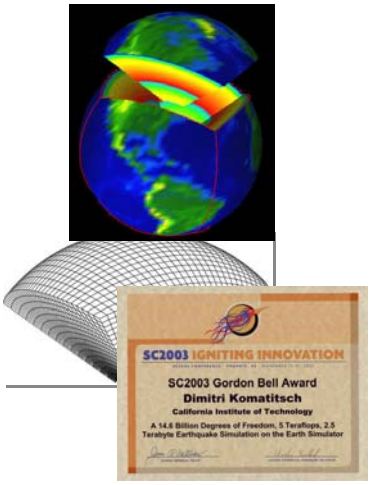
Load balance



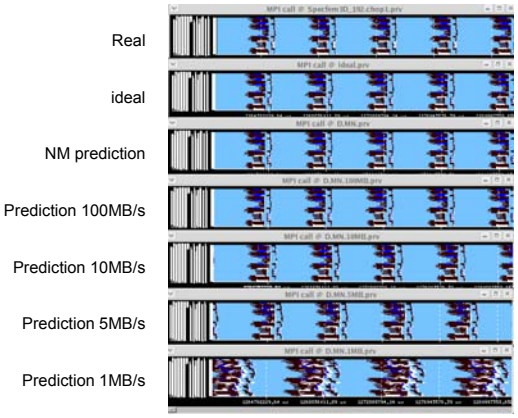
- Often small (~ 10%, programmers tried to) but sometimes large
- Causes:
  - Intrinsic to the algorithm/problem (Phased / time varying behavior, Data dependent computational load / access pattern)
  - Caused by resources (Processor heterogeneity in a chip/board, OS noise/user daemons,…)
- Programmer
  - Often unaware
  - Can improve (“hand optimized” schedules) at high cost.
- Often bad for real “Real apps”

Example: Specfem3D

- Should I introduce asynchronous communication?



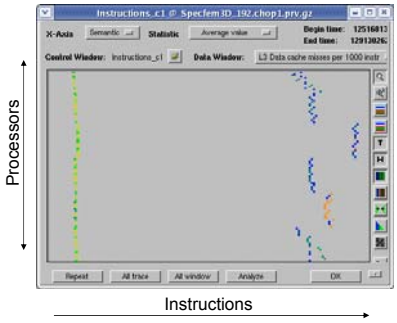
Courtesy Dimitri Komatitsch



Specfem3D

- Load Balance? Instructions and cache misses

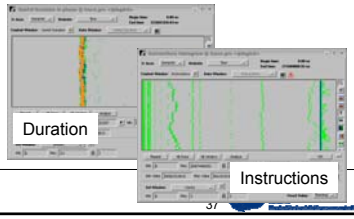
@ 96 processors



Color: cache misses

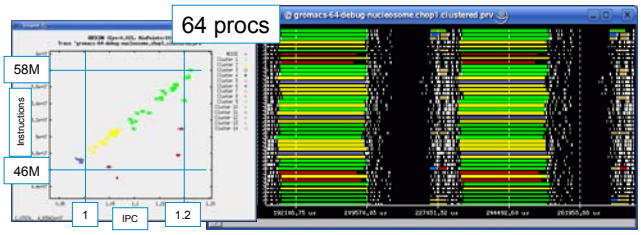
- Even more
- many
- Few

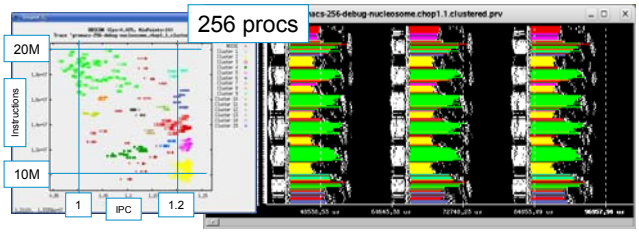
- Work on domain decomposition, element numbering



Gromacs: Particles interaction

- Load imbalance
  - Appears when P Grows
  - Nature
    - Computation
    - IPC
- Changes shape
  - Trapezoidal
  - IPC ∝ Instr

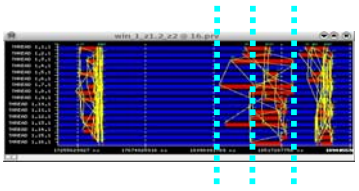




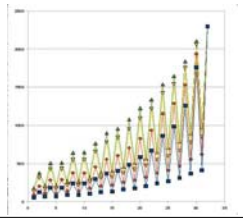
  - 2x instr
  - 20% IPC
  - IPC ∝ 1/Instr

Load balance and power saving

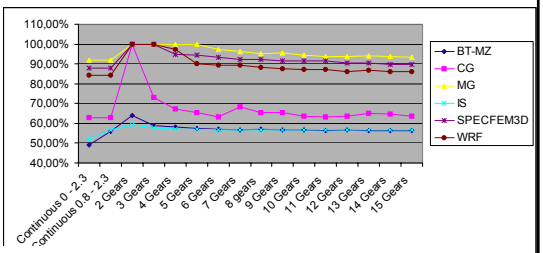
- Potential of DVFS
  - Static for each core (0.8 – 2.3 GHz)
  - Dimemas + balance computation + power models + performance models
- Observations
  - Few Gears (~6 ?)
  - Lower limit may restrict gain for some applications
  - …



MIN AVG MAX

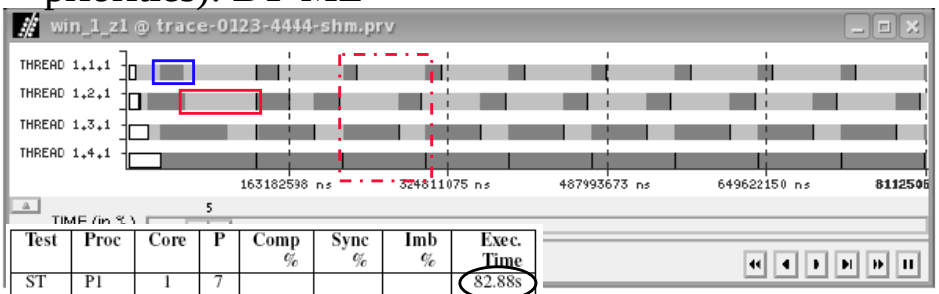


0.3
0.5
0.8
1



M. Etinski, et al, "Power-Aware Load Balance Of Large Scale MPI Applications "

Dynamic load balance (POWER5 hardware priorities): BT-MZ



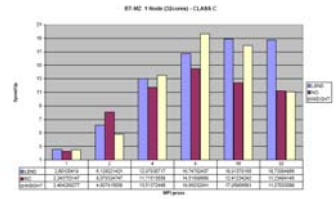
Test	Proc	Core	P	Comp %	Sync %	Imb %	Exec. Time
ST	P1	1	7				82.88s
	P2	2	7				
A	P1	1	4	17.63	82.32	82.23	81.64s
	P2	1	4	28.91	71.02		
	P3	2	4	66.47	33.4		
	P4	2	4	99.72	0.09		
B	P1	1	3	52.33	47.49	70.93	127.91s
	P2	2	3	99.64	0.14		
	P3	2	6	28.87	71.07		
	P4	1	6	46.26	53.65		
C	P1	1	4	65.32	34.48	45.99	75.62s
	P2	2	4	99.68	0.12		
	P3	2	6	53.78	46.11		
	P4	1	6	85.88	14.44		
D	P1	1	4	82.73	17.10	33.38	66.88s
	P2	2	4	73.68	26.17		
	P3	2	5	66.40	33.47		
	P4	1	6	99.72	0.09		

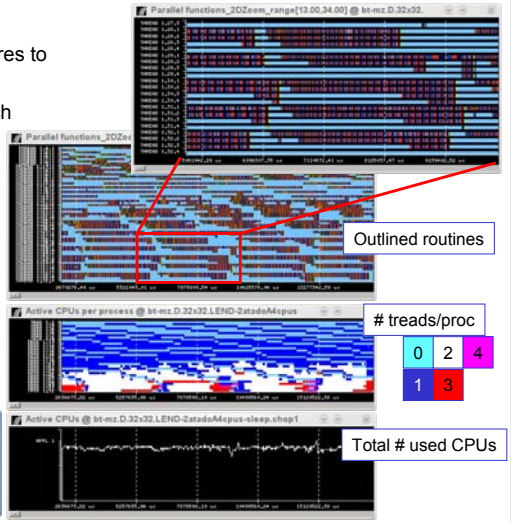
- Imbalanced application
  - "similar" demand for different amount of time
- Different mappings and priorities
- Potential but also risk !!!!

C. Bonetti et al. "A user level load and resource balancer for HPC applications"

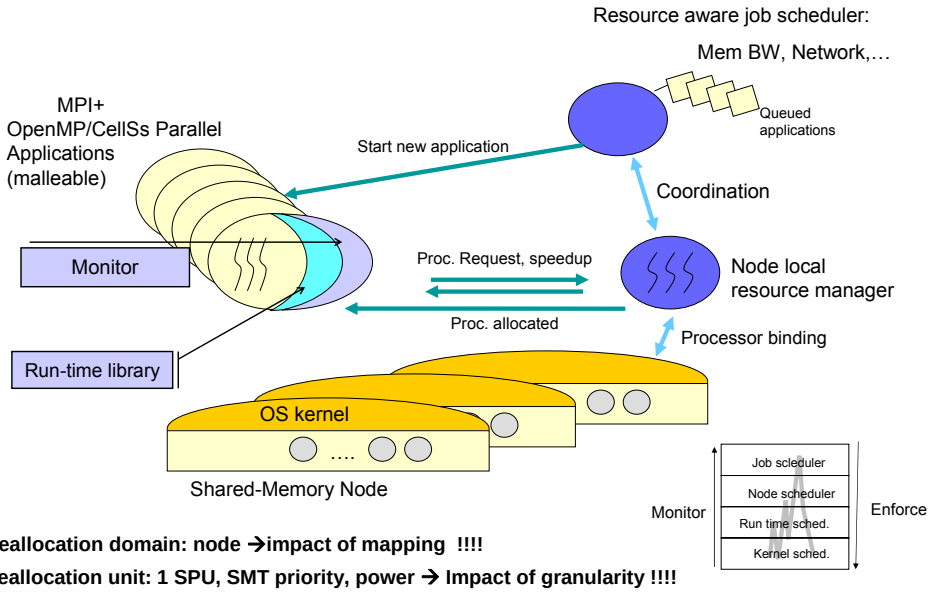
Dynamic Load Balancing run time

- NAS BT-MZ @ Power6 32 way SMP
- MPI + OpenMP (StarSs)
- Processes blocking at MPI calls lend cores to other processes
- Job is started with P MPI processes each with 1 OpenMP thread
- Memory issues !!!!!





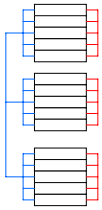
Malleability and Resource Management



Reallocation domain: node →impact of mapping !!!!

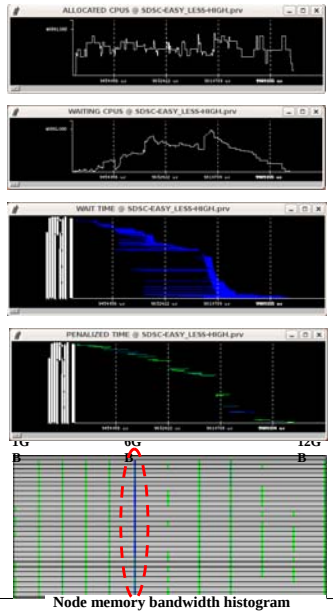
Reallocation unit: 1 SPU, SMT priority, power → Impact of granularity !!!!

Resource aware scheduling policies

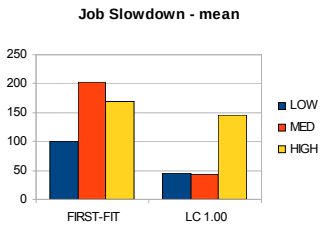


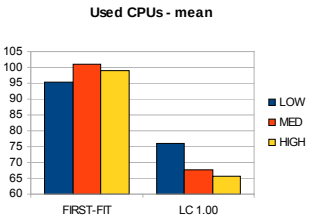
- Can resource aware policies result in better efficiency?
  - Alvio simulator:
    - Reservation tables: Dynamic
    - Job scheduling and resource selection policies
    - Sharing penalty models: ie: node memory BW
- Resource selection policy: Less Consume
  - Allocate nodes that “minimize” penalty (i.e. node bandwidth oversubscription)
- Power Aware Scheduling
  - Interaction between DVFS and Job scheduling
- QoS metrics
  - Definition of metrics
  - “RT” policies

Resource aware scheduling policies

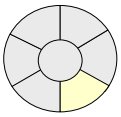


Node memory bandwidth histogram





Node Architecture

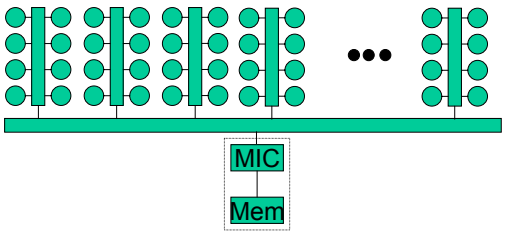


Infrastructure to study multicore architectures

- Started with an execution driven Cell simulator.
  - Tooooooooo slow.
- Tasksim: coarse grain trace driven simulation. Focus on concurrency and memory subsystem
  - Input
    - Paraver traces generated by CellSs (in progress by SMPSs) run time
    - Computation bursts, DMA requests, dependences
  - Simulator
    - Scale computation bursts (independent for PPE and SPE)
    - Simplified EIB
    - MIC
    - Simple memory module, Detailed DDRx module
  - Output
    - Statistics
    - Paraver traces

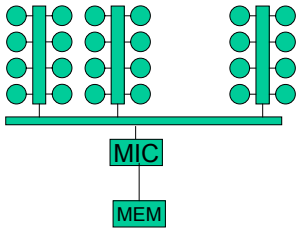
TaskSim: multicore scalability

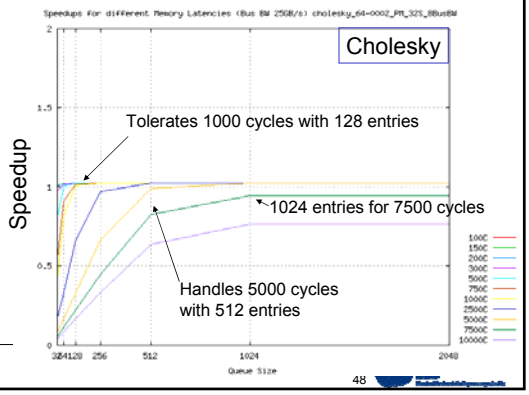
- Scalability with # SPEs



TaskSim: Latency Tolerance

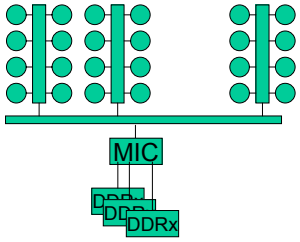
- Ideal Memory:
  - 128 bytes/cycle
  - Different latencies
- It is possible to tolerate huge latencies
- Required MIC entries for in flight accesses



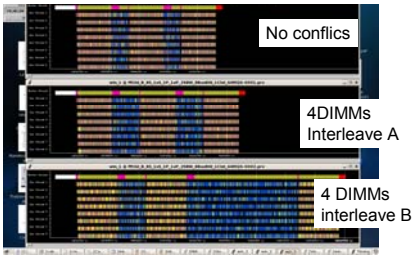


TaskSim: predicting memory performance

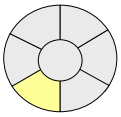
- Impact of DRAM interleave



25.6 GB/s in all cases

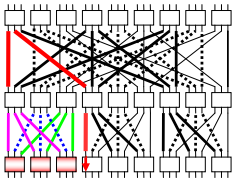


Interconnect

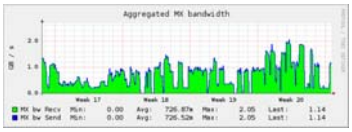


Interconnects

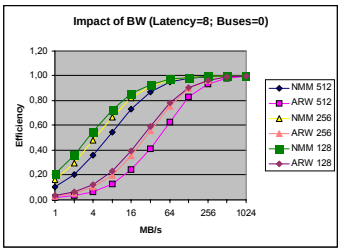
- Forecasts: significant percentage of total power at Exaflop scales.
- An over dimensioned resource?



  - Marenostrum interconnect usage

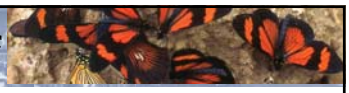


  - Application sensitivity to interconnect parameters



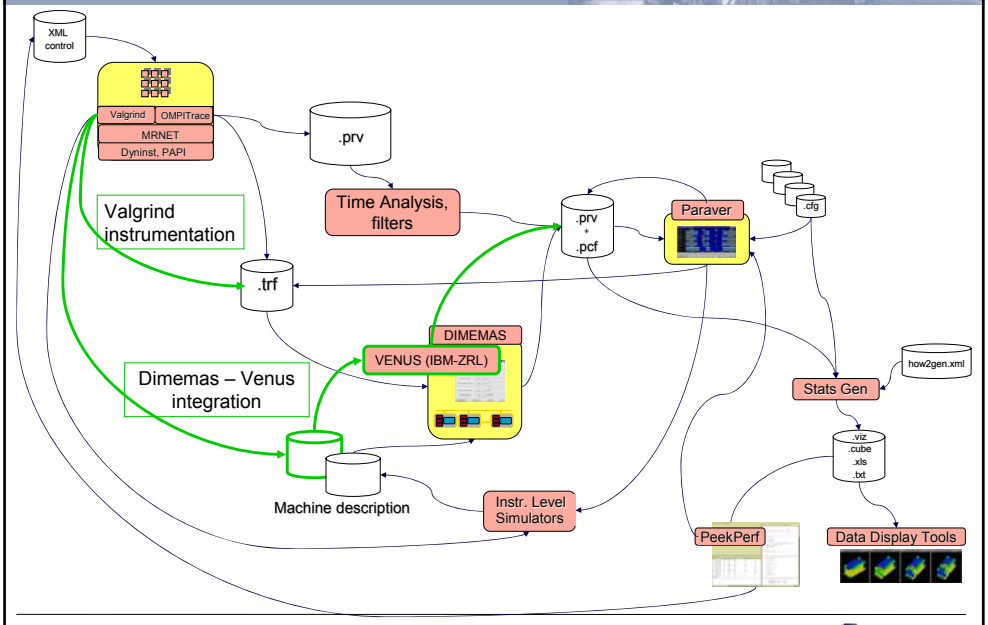
- Revisit interconnect needs, behavior,…
  - Which are the most important factors?
  - Bandwidth? Topology? hardware/software?

Revisit interconnect … a valley of



- Application
  - Communication – computation overlap
- Protocol issues
  - Control messages, segments,…
  - Often overlooked!!!!
- Topology & routing
  - Need for “non blocking” networks? Impact of slimming? In performance? In cost?
  - Static routing. Is it really bad? why?
  - dynamic routing. Is it really Needed? How good is a local dynamic routing?
  - Oblivious vs. pattern aware?
  - Contention
    - Network or injection contention
    - Internal vs external contention. Which is more important? When is external contention harmful,….
- Switch and adapter

Evaluation infrastructure



Butterflies !!!!!!!

Software: Application & MPI library
Protocol/Middleware
Network topology & routing
Switch & adapter technology

WRF - 256

Infinite CPU speed, input-queued switch

Arbitration choices in switch

All2all - 32

Eager Protocol

no imbalance

No network contention !!!!

Issues addressable at NIC level !!!!!

1µs imbalance → 1.5 ms

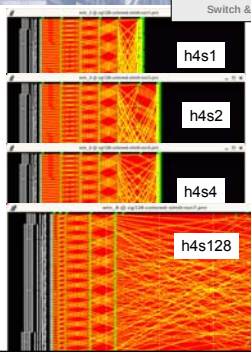
Protocol /data messages interaction in adapter

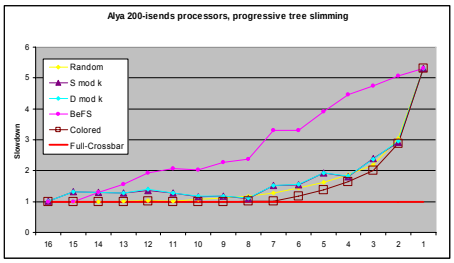
Order of request arrivals
Endpoint contention
propagation

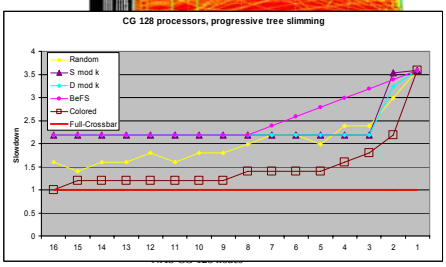
Rendez-vous protocol – 1us imbal. In a thread

Static Source Based Routing

- Random: Not good
- Oblivious: quite OK. Some “pathologic” cases
- Slimmed trees
  - fair tolerance to small levels
- Pattern aware
  - avoid “pathologies”
  - Tolerate high levels of slimming





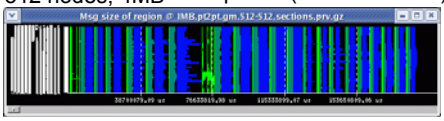


Contention impact

- Real traces
- In multi-user environments

Dependence on appl. phase (comm. Pattern)

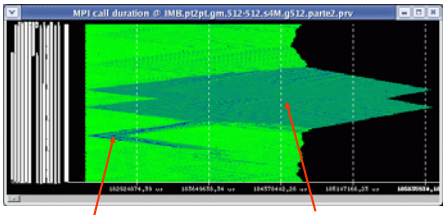
512 nodes, 4MB



64 nodes, G=8, 4MB

External contention

Internal contention



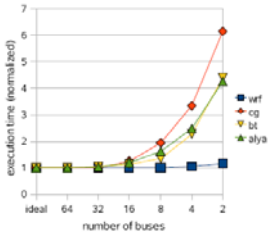
Bubble propagation

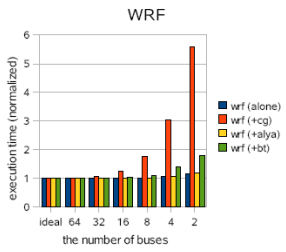
Propagation of internal contention

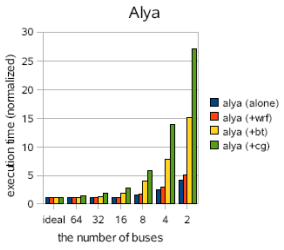
What is the benchmark measuring?
Appropriate number of iterations?

External vs. internal contention

- Internal contention
  - Not very important for some applications
- External contention
  - Some applications can significantly hurt others
  - > multiplicative effect



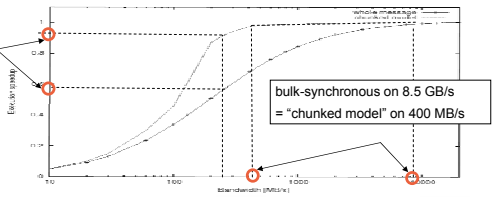




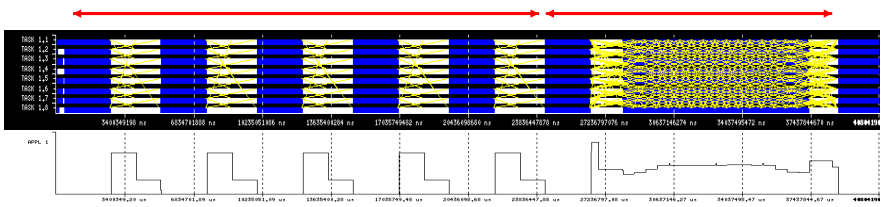
...mication computation overlap

with network bandwidth 250 MB/s
"chunked model" 1.62 time faster

bulk-synchronous on 8.5 GB/s
= "chunked model" on 400 MB/s



Link Bandwidth = 250 MB/s



* communication pattern: two sided ring

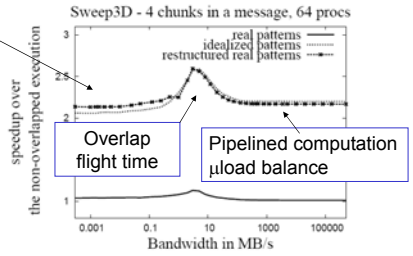
V. Subotic et al. "Overlapping communication and computation by enforcing speculative data-flow"

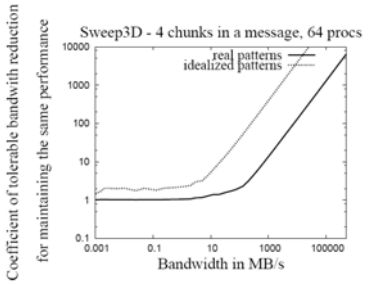
computation

- Potential:
  - Speedup
  - Use slower networks!!
- Overlap comm - comp :
  - Real production/consumption patterns
    - Limited speedup
  - Restructured
    - Predictions: Fair – Good
    - Need to restructure

Pipelined data transfer

Possible serialization issues





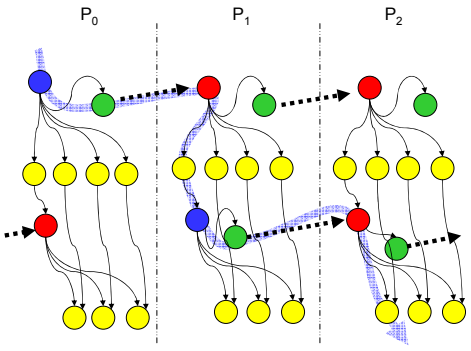
V. Subotic et al. "On the potential of automatic computation – communication overlap"

MPI + SMPSs: the linpack example

- Overlap communication/computation
- Extend asynchronous data-flow execution to outer level
- Automatic lookahead

```
…
for (k=0; k<N; k++) {
   if (mine) {
      Factor_panel(A[k]);
      send (A[k])
   } else {
      receive (A[k]);
      if (necessary) resend (A[k]);
   }
   for (j=k+1; j<N; j++)
      update (A[k], A[j]);
…
```

```
#pragma css task inout(A[SIZE])
void Factor_panel(float *A);
#pragma css task input(A[SIZE]) inout(B[SIZE])
void update(float *A, float *B);
```

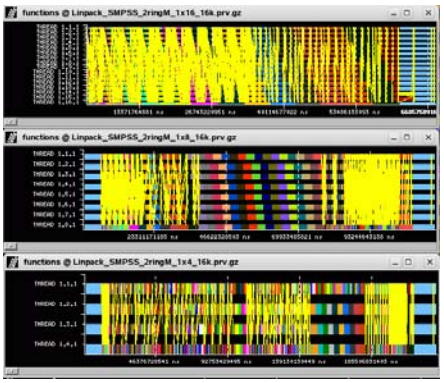



```
#pragma css task input(A[SIZE])
void send(float *A);

void receive(float *A);
#pragma css task input(A[SIZE])
void resend(float *A);
```

MPI + SMPSs: the linpack example

- Overlap communication/computation
- Extend asynchronous data-flow execution to outer level
- Automatic lookahead



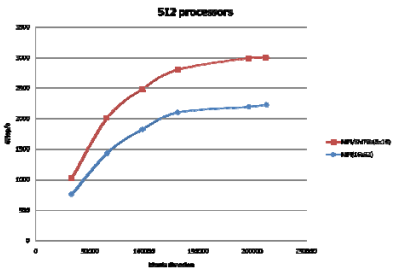
```
…
for (k=0; k<N; k++) {
   if (mine) {
      Factor_panel(A[k]);
      send (A[k])
   } else {
      receive (A[k]);
      if (necessary) resend (A[k]);
   }
   for (j=k+1; j<N; j++)
      update (A[k], A[j]);
…
```

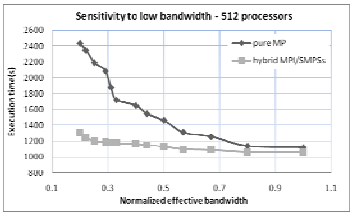
```
#pragma css task input(A[SIZE])
void send(float *A);
#pragma css task output(A[SIZE])
void receive(float *A);
#pragma css task input(A[SIZE])
void resend(float *A);
```

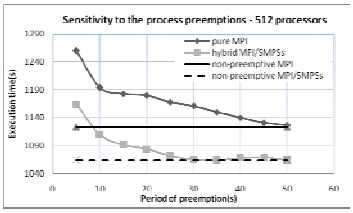
```
#pragma css task input(A[SIZE])
void Factor_panel(float *A);
#pragma css task input(A[SIZE]) inout(B[SIZE])
void update(float *A, float *B);
```


MPI + SMPSs: the linpack example

- Performance
  - Higher at smaller problem sizes
  - Improved Load balance (less processes)
  - Higher IPC
  - Overlap communication/computation
- Tolerance to bandwidth and OS noise

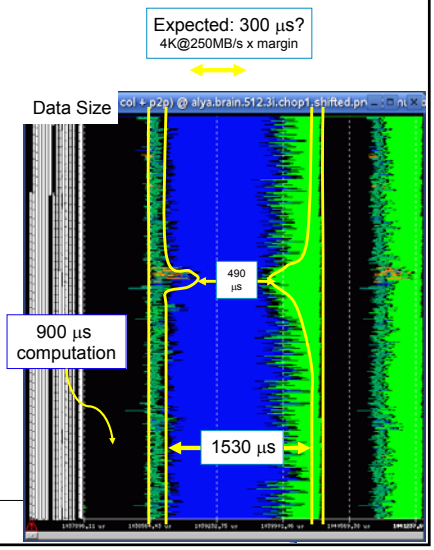




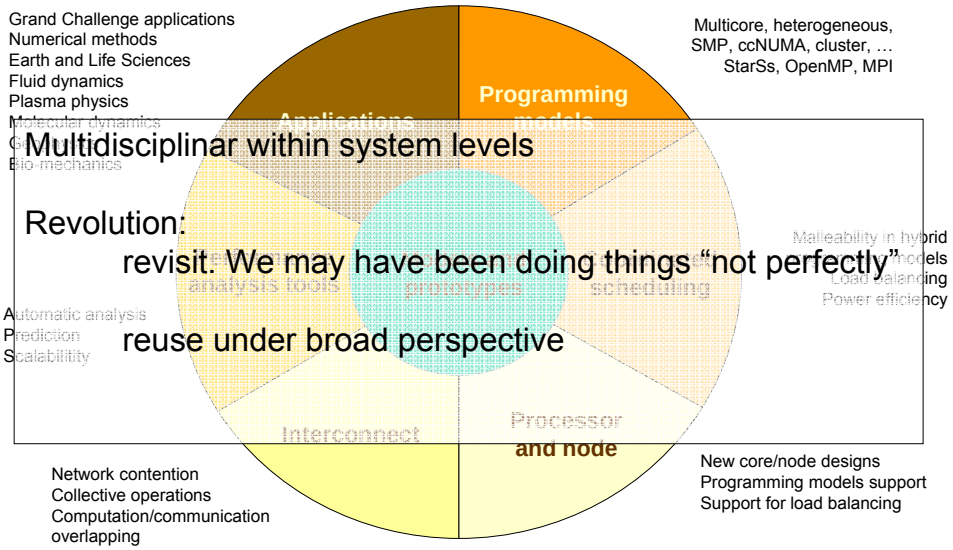


An example of holistic issues: brain simulation with Alya

- Algorithm with optimized computational cost
  - All_reduce data size = cst (not ∝1/P): Pays off? Alternative??
- Very fine granularity
  - Scale to 10K cores? Problem size?
- Impact of All_reduce performance
  - Variance
    - Network contention / adapter / protocol issues
  - Global synchronization
    - Asynchronism/overlap potential? No?
- Load Balance
  - Computation: ~fair
  - Point to point communication: BAD
- Hybrid (MPI+OpenMP/…)?
  - Shorter collective (less nodes)
  - Opportunity to load balance

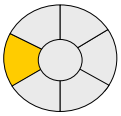


Our holistic view

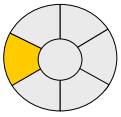


Analysis of applications

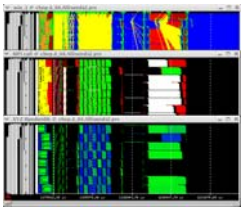
Performance tools



Performance tools (WP3)



- Issues:
  - Scalability, interoperability, intelligence
  - Reduce analysis cost, maximize quality of information
- Topics:
  - Spectral analysis techniques
  - Clustering
  - Sampling + tracing
  - Dynamic range: microarchitecture ↔ large runs @ large core counts
  - Instrumentation on P7, BG/Q
  - Integration to Peekperf/Eclipse
  - Production monitoring and data base

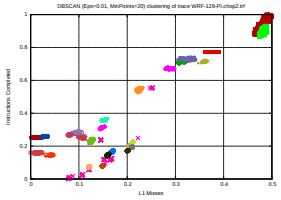


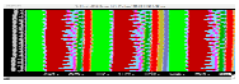
Understanding performance

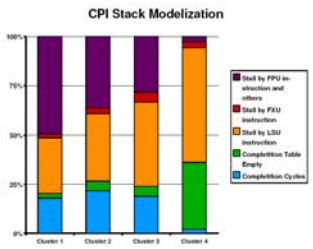
- Today's practice:
  - Mostly coarse grain measure
    - … but often the importance is in the details (butterfly effect)
  - Little modeling
- We need:
  - Insight, not data.
  - More intelligence
    - To maximize the ratio information/data emitted by our tools.
    - To focus on the relevant issues.
  - Including modeling
    - To explain what happens and what could happen.

CEPBA-Tools Environment

- Intelligence
  - Signal processing
  - Clustering
  - Models
  - ….
- Moving to online
  - MRNET infrastructure
- Integration with other tools







Region	IPC	L3D misses per 1000 instr	D TLB misses per 1000 instr	L1D $ misses per 1000 instr	Bytes / Instr
1	0,57	2,34	0,01	75,55	0,30
2	0,54	0,48	0,05	52,6	0,06
3	0,53	1,18	0,14	47,64	0,15
4	0,62	0,38	0,04	43,27	0,05
5	0,42	1,56	0,18	43,84	0,20