

Targeting Multi-Core systems in Linear Algebra applications

Alfredo Buttari, Jack Dongarra, Jakub Kurzak and Piotr Luszczek

Multi-cores / Many-cores 2007

The free lunch is over

Problem

- power consumption
- heat dissipation
- pins

Solution

reduce clock and
increase execution
units = Multicore

Result

Non-parallel software won't run any faster. A new approach to programming is required.

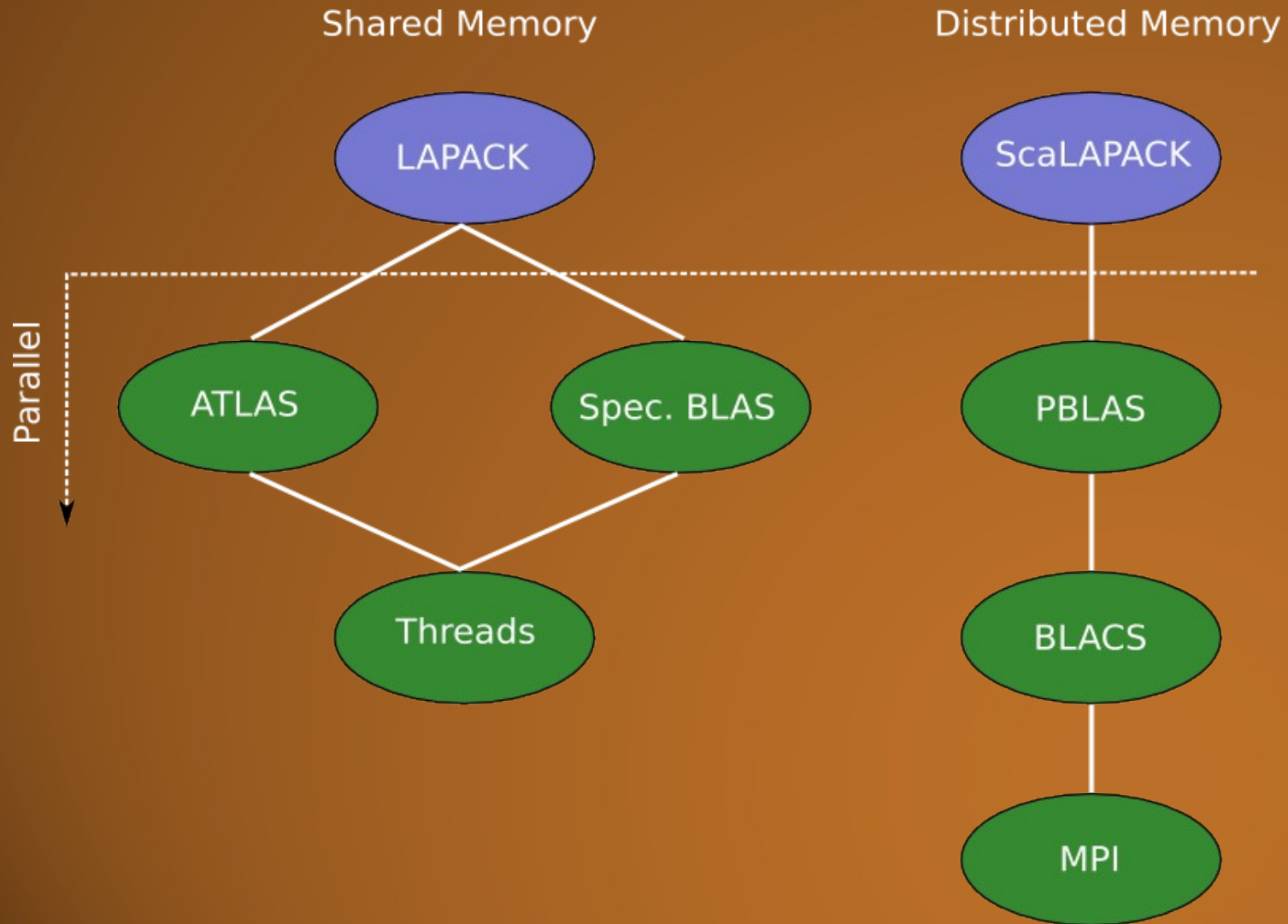
What is a Multicore processor, BTW?

“a processor that combines two or more independent processors into a single package” (wikipedia)

What about:

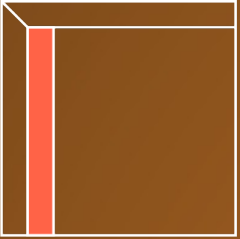
- types of core?
 - homogeneous (AMD Opteron, Intel Woodcrest...)
 - heterogeneous (STI Cell, Sun Niagara...)
- memory?
 - how is it arranged?
- bus?
 - is it going to be fast enough?
- cache?
 - shared? (Intel/AMD)
 - non present at all? (STI Cell)
- communications?

Parallelism in Linear Algebra software

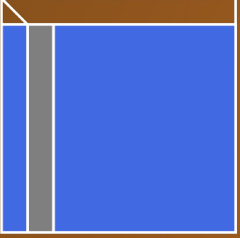


Parallelism in LAPACK: LU factorization

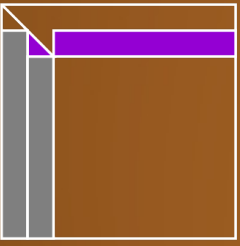
DGETF2



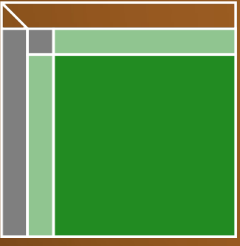
DLSWP



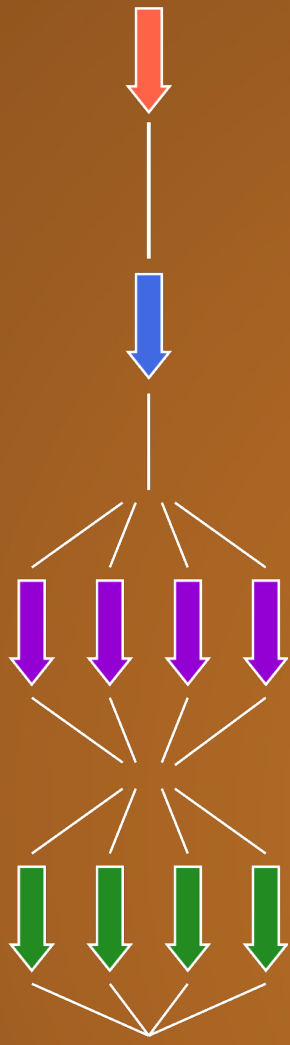
DTRSM



DGEMM



BLAS2 operations cannot be efficiently parallelized because they are bandwidth bound.



- strict synchronizations
- poor parallelism
- poor scalability

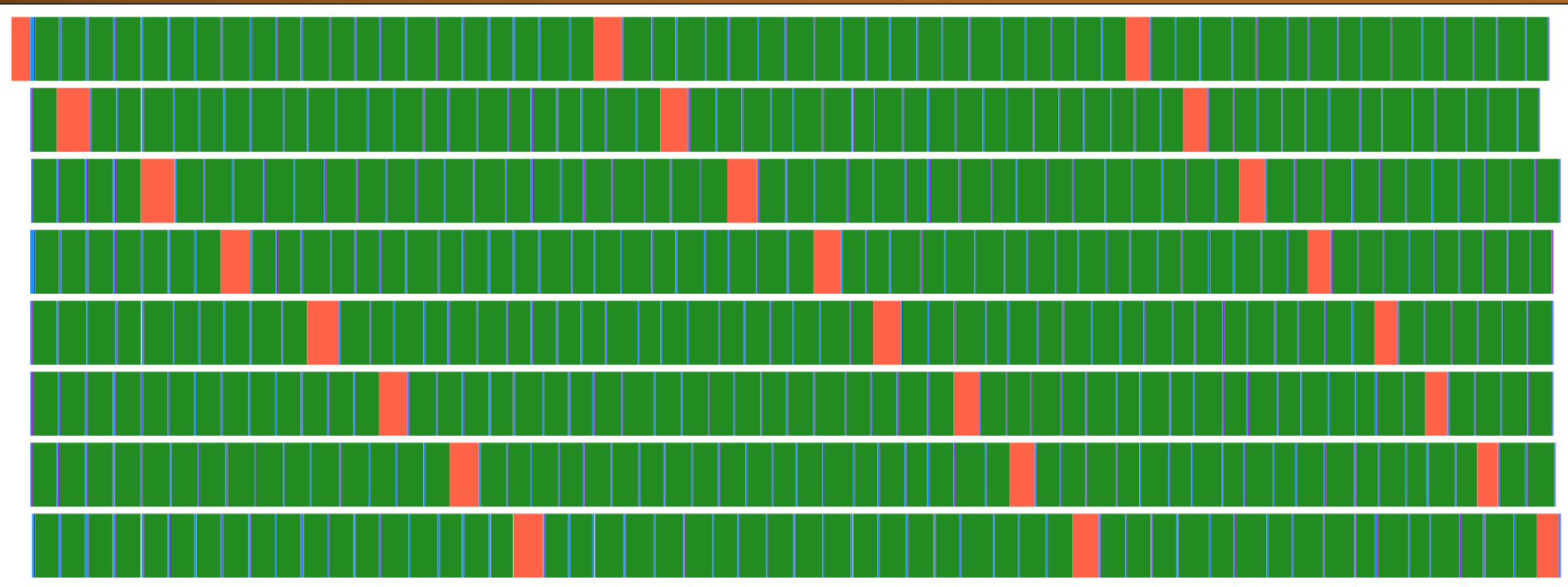
Parallelism in LAPACK: LU factorization

The execution flow if filled with stalls due to synchronizations and sequential operations.



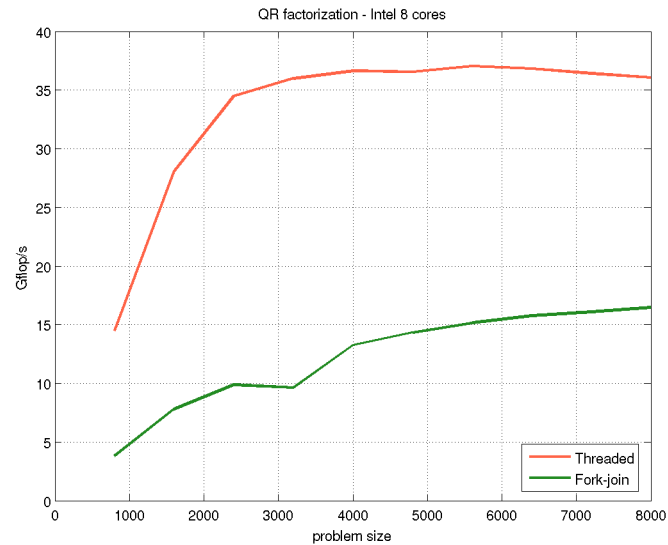
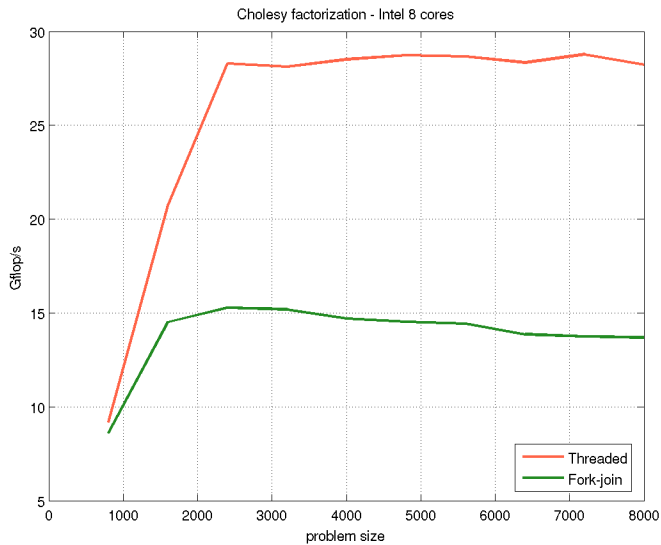
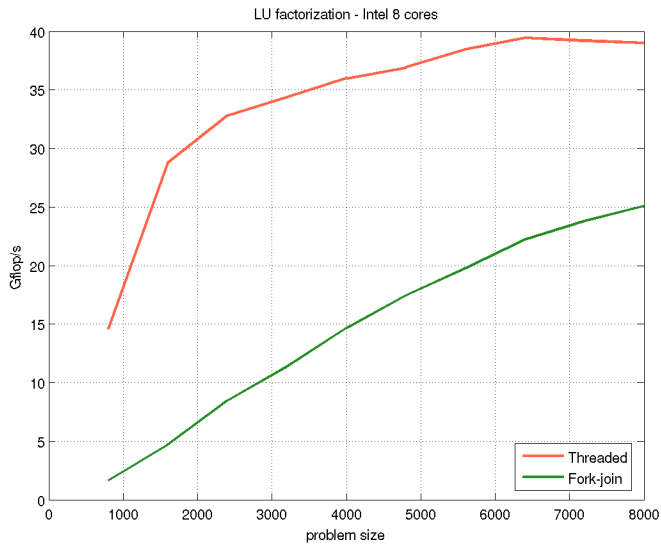
Parallelism in LAPACK: LU factorization

- higher flexibility
- some degree of adaptativity
- no idle time
- better scalability

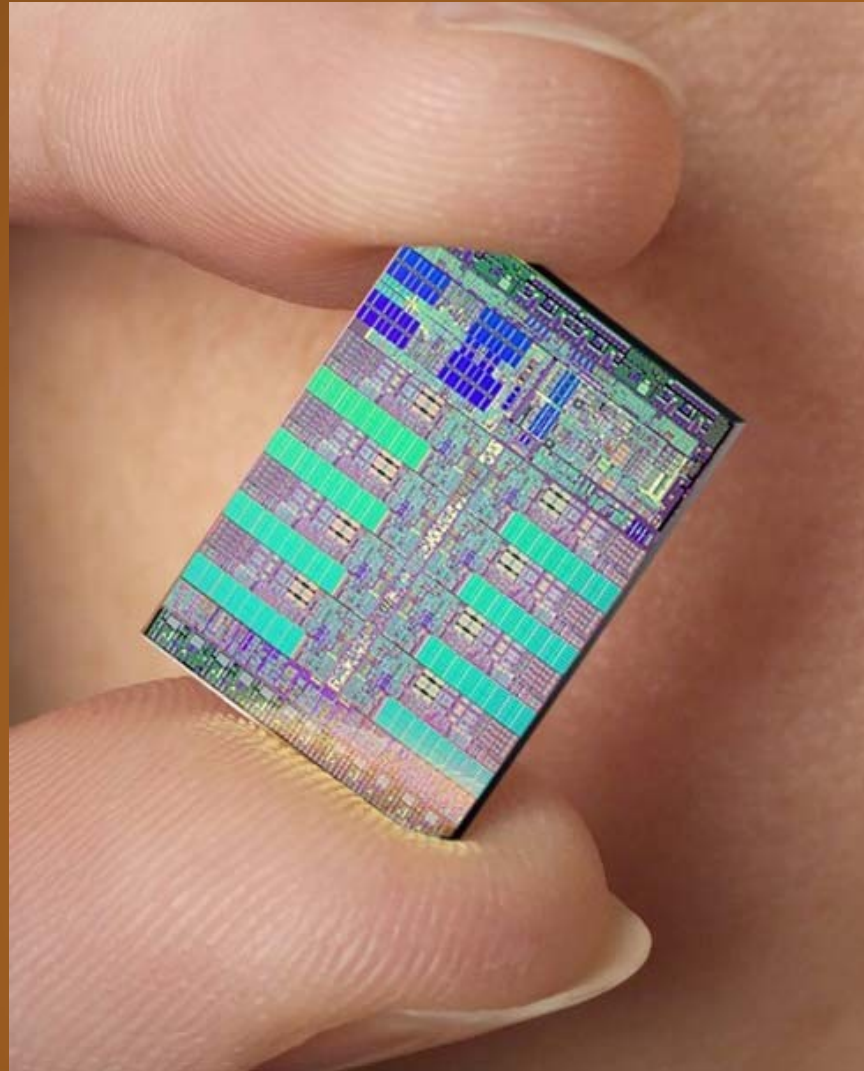


Parallelism in LAPACK: results

Performance results on an Intel 2x4 cores machine.



Exploring the Cell processor



Iterative refinement: once upon a time...

Iterative refinement has been developed as a method to improve solution accuracy.

First solution is computed and then correction are computed and applied in order to improve accuracy.

$$(1) x_0 \leftarrow A^{-1} b$$

until convergence **do**

$$(2) \quad r_k \leftarrow b - Ax_{k-1}$$

$$(3) \quad z_k \leftarrow A^{-1} r_k$$

$$(4) \quad x_k \leftarrow x_{k-1} + z_k$$

done

Iterative refinement for speed

If the most expensive computations are performed in SP while the cheap part of the iterative process is done in DP we can:

- Have a solution that is DP accurate
- Run very close to the speed of SP

$$(1) x_0 \leftarrow A^{-1} b$$

until convergence **do**

$$(2) r_k \leftarrow b - Ax_{k-1}$$

$$(3) z_k \leftarrow A^{-1} r_k$$

$$(4) x_k \leftarrow x_{k-1} + z_k$$

done

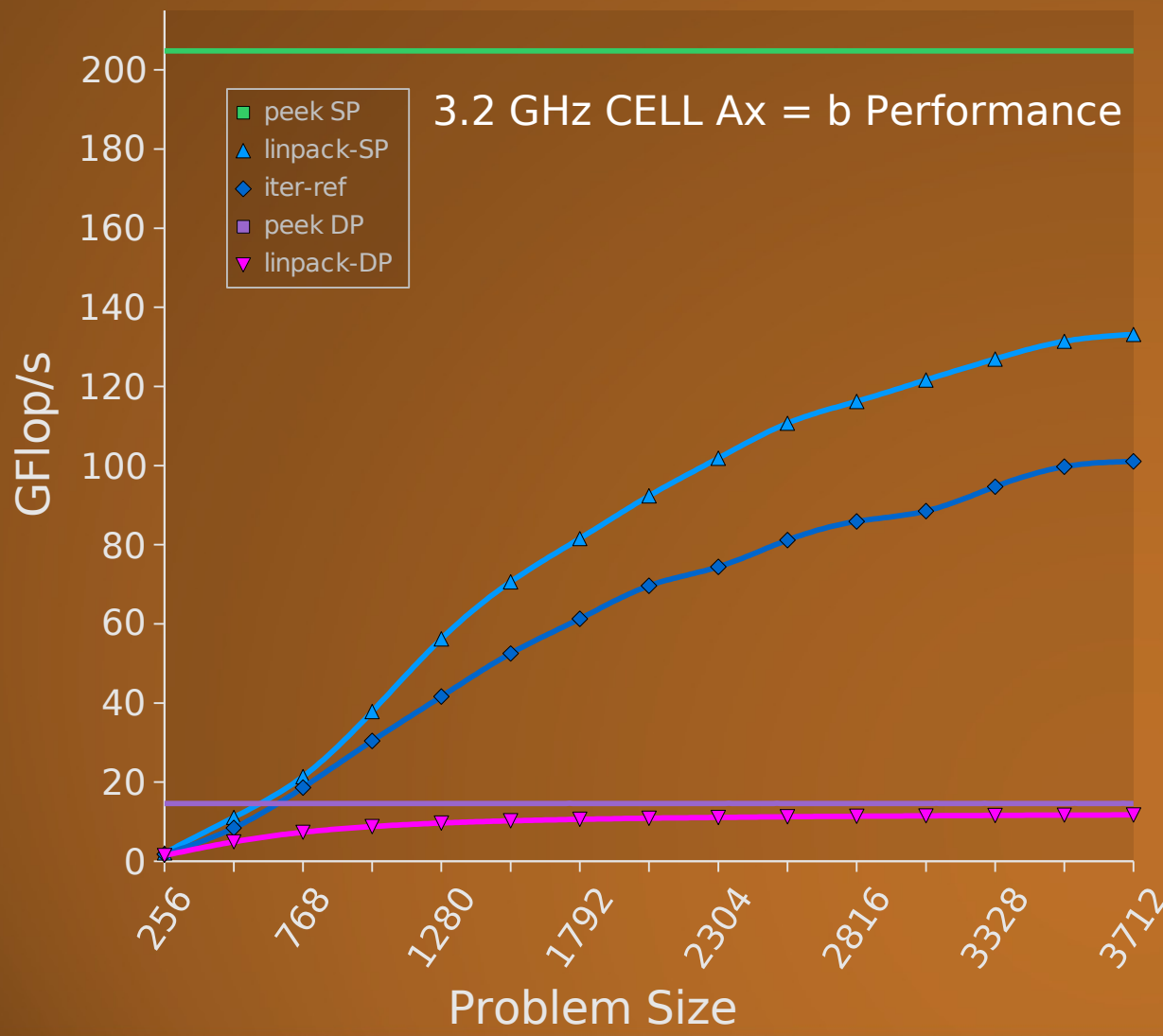
■ single precision

■ double precision

Iterative refinement on the Cell: LU

- LAPACK FORTRAN 77 DSGESV on top
 - linpack-SP
 - SGETRF
 - SGETRS
 - Additional SPE-parallel code
 - Conversion from standard to block layout
 - Conversion from single to double precision
 - DLANGE – matrix norm (DP)
 - DGEMM – matrix multiply (DP)
 - PPU auxiliary Level 1 BLAS (DAXPY, DLACPY, DNRM2)
-
- Block data layout (64×64 SP, 32×32 DP)
- Large TLB Pages

Iterative refinement on the Cell: LU



Iterative refinement on the Cell: LU

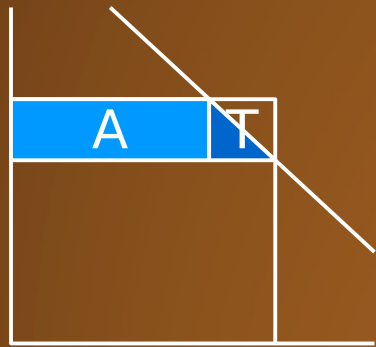
8/1/2006

75

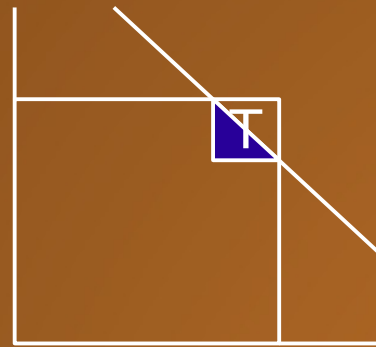
Computer (Full Precision)	Number of Procs or Cores	R_{max} GFlop/s	N_{max} Order	$N_{1/2}$ Order	R_{Peak} GFlop/s
IBM SP (375 MHz POWER3)	88	99.7	88000		132.0
SGI Origin 2000 250/300 MHz Cluster (2x64x250+2x64x300)	256	98.87	81920	81920	140.8
Sun Fire 6900 (UltraSPARC IV, 1.2 GHz)	48	98.26	96116	8300	115.2
IBM Cell BE (3.2 GHz)*****	9	98.05	4096	1536	14.6 (64 bit) 204.8 (32 bit)
SGI Altix 3000, 900 MHz	32	97.67	82079	82079	115
Kepler (192 PIII@650 MHz + 4 PIII@733 MHz)	196	96.25	109760	12320	127.7
IBM SP (375 MHz POWER3)	84	95.5	88000		126.0
IBM eServer pSeries 690 Turbo(1.3 GHz Power 4)	32	95.26	108000	7000	166.4
Cray X-1 (800 MHz)	8	95.2	61440	5632	102.4
Fujitsu VPP700/46 (7nsec)	46	94.3	100280	8280	101
SGI Origin 300 (500 MHz, w/Myrinet)	128	94.15	81920	81920	128
HP 9000 rp8420-32 (1000MHz PA-8800)	32	94.1	58960	5200	128
Sun Fire 15K (1050MHz/8MB E\$)	56	94.06	96116	10000	117.6
IBM S80s (450 MHz, SP switch)	192	93.87	82000	21000	173
ClearSpeed CSX600 Advance accelerator boards (dual boards each with 96 cores at 250 MHz) (frontend HP ProLiant DL380 G5, dual node Intel Xeon 5100 dual core, 3 GHz)	200	93.3	45000		240

Iterative refinement on the Cell: Cholesky

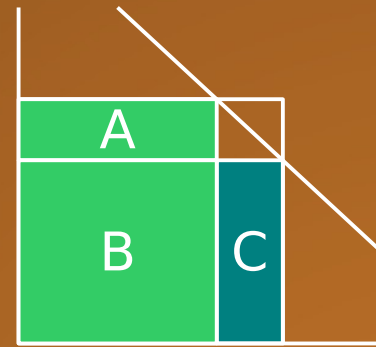
$T = T - A \times A^T$
SYRK



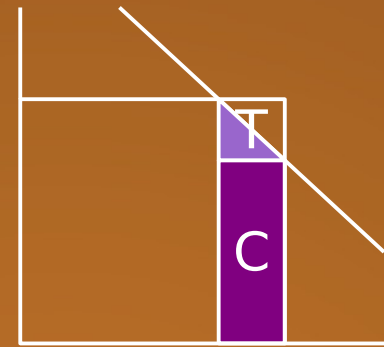
$T = LL^T$
POTRF



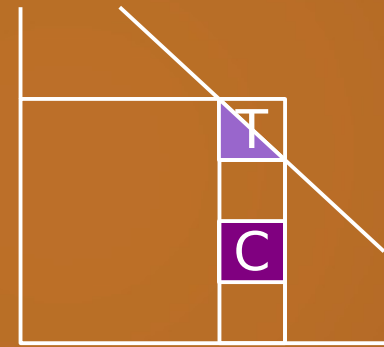
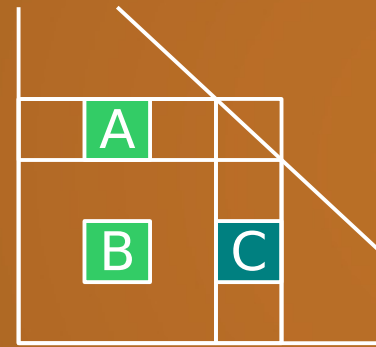
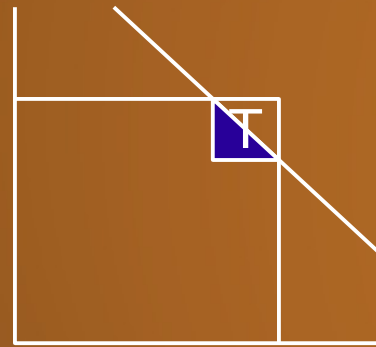
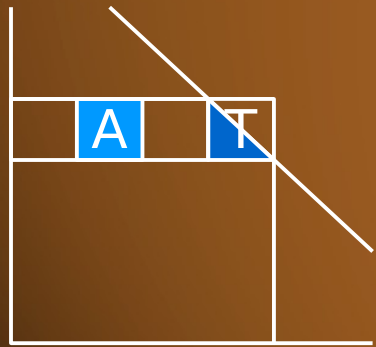
$C = C - B \times A^T$
GEMM



$C = C \setminus T$
TRSM

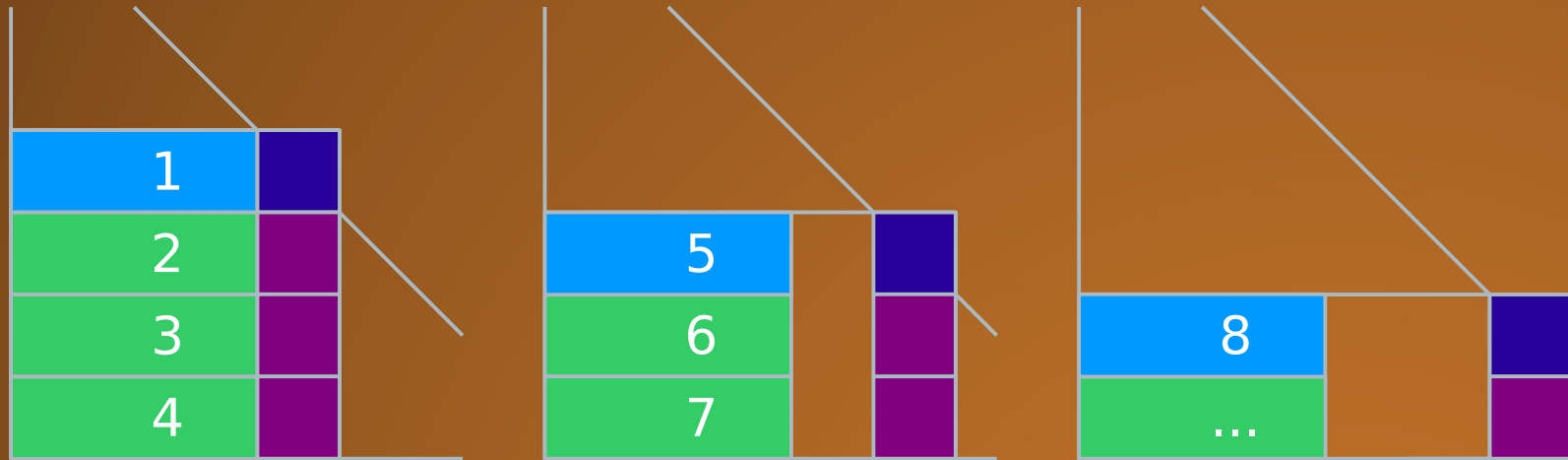


2-D blocking with 1-D distribution



Iterative refinement on the Cell: Cholesky

Pipelining Loop Iterations



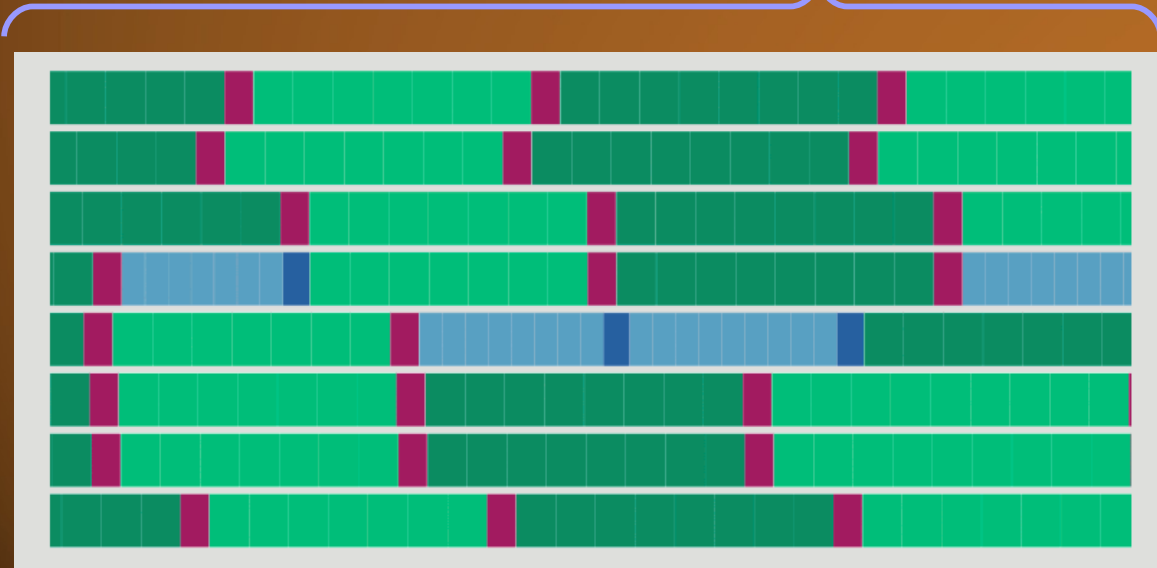
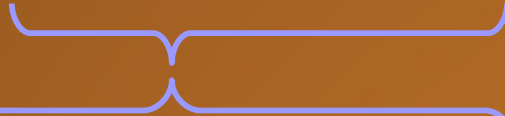
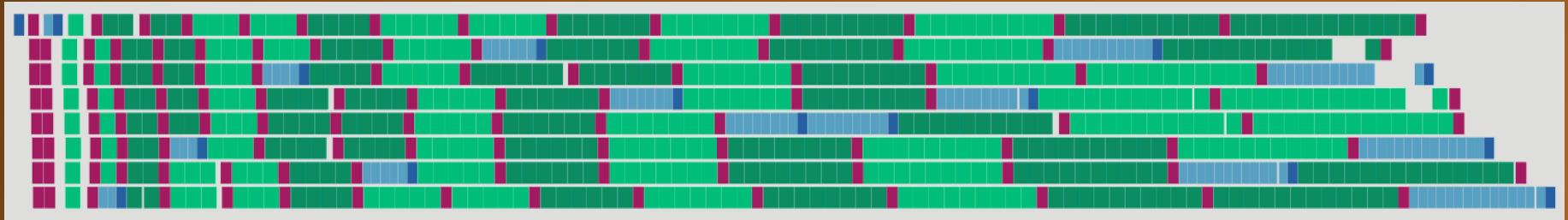
1D Work Partitioning

- Facilitates data reuse,
- Prevents bus saturation.

2D Dependency Tracking

- Facilitates iteration pipelining,
- Eliminates load imbalance.

Iterative refinement on the Cell: Cholesky



Pipelining:

- Between loop iterations.

Double Buffering:

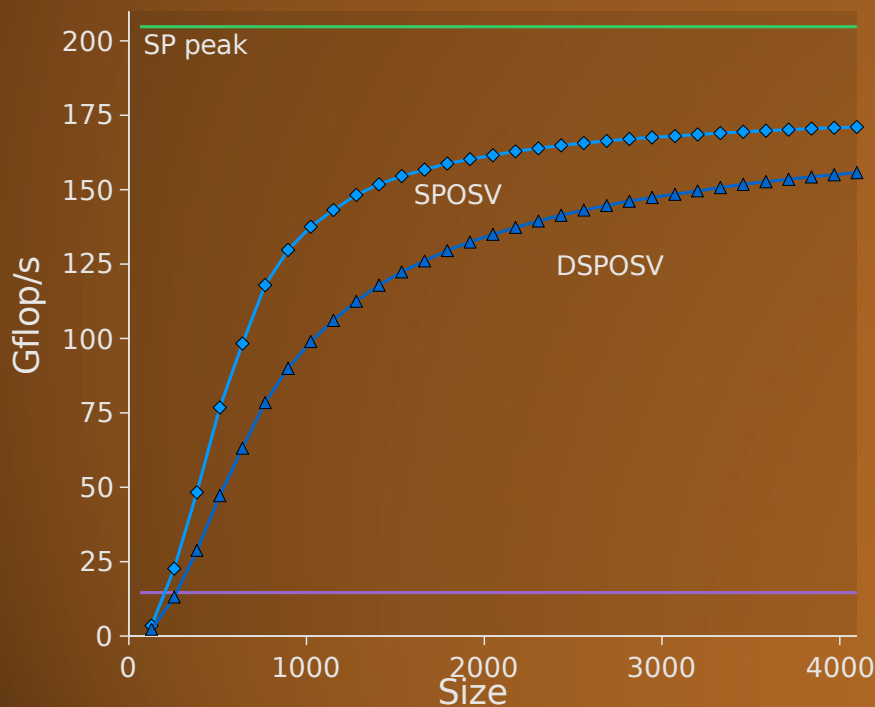
- Within BLAS,
- Between BLAS,
- Between loop iterations.

Result:

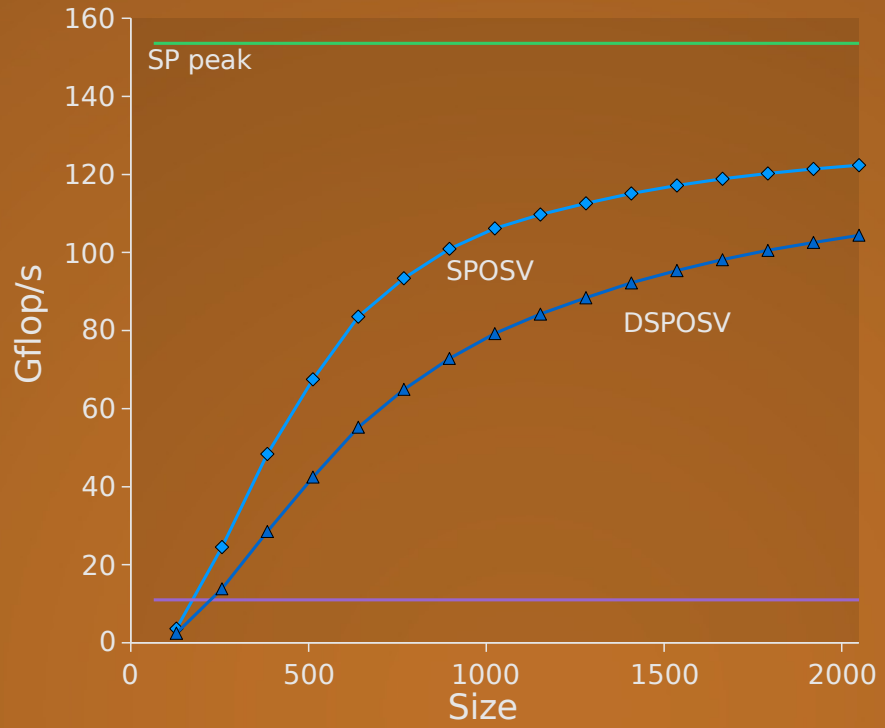
- Minimum load imbalance,
- Minimum dependency stalls,
- Minimum memory stalls (no waiting for data).

Iterative refinement on the Cell: Cholesky

Cell Blade



PS3



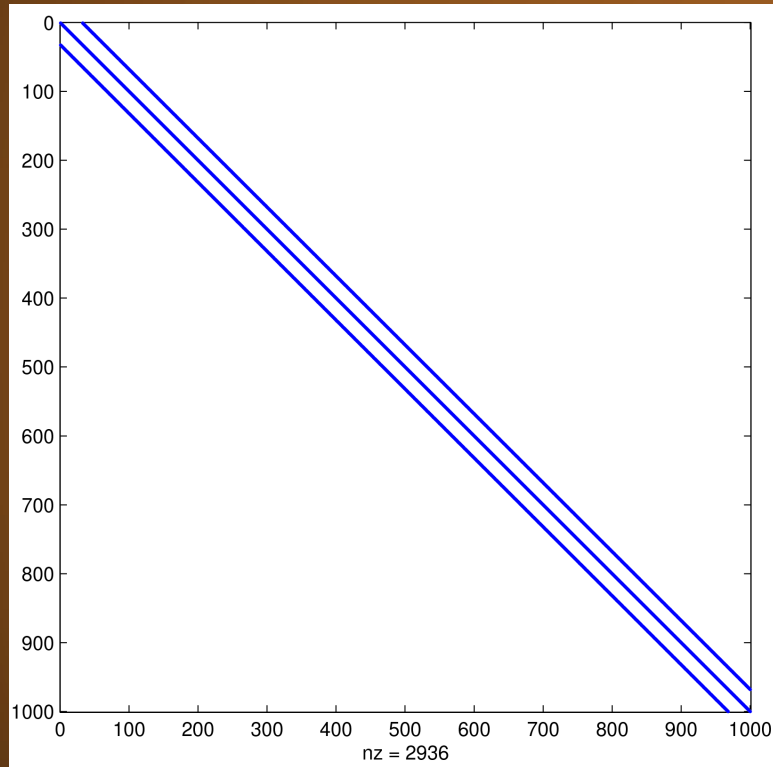
Sparse LA on the Cell

Why you can't obtain performance on sparse operations (in the general case):

- exploit regular memory banks access
 - data is “sparse” and can be accessed only with indirect addressing
- fetch big chunks of data from GS to reduce DMA latency
 - impossible on the source vector if no tiling
- double buffering
 - no surface to volume so communications are more expensive than computations
- vectorization:
 - JAD format and Segmented Scan: no gather operations on SPEs
 - BCSR: pay the cost of fill-in on 4x4 blocks

Sparse LA on the Cell

One lucky case:



the goods:

- stride-1 access on the source vector
- easy vectorization
- regular memory access pattern
- big chunks of data may be fetched at once

the bad:

- still no surface to volume

Upper bound is bus speed if no reuse.

Sparse LA on the Cell

$$r_0 = b - Ax_0$$

$$z_1 = rM^{-1}$$

$$\rho_1 = \langle r, z \rangle$$

$$9n/7n = 1.28$$

$$5.842$$

$$\text{Gflops} = 18 \text{ GB/s}$$

$$q_1 = Az_1$$

$$\alpha_1 = \rho_1 / \langle z, q \rangle$$

$$8n/4n = 2.00$$

$$10.653$$

$$\text{Gflops} = 20 \text{ GB/s}$$

$$x_1 = x_0 + \alpha_1 z_1$$

$$\text{normx} = \|x_1\|$$

$$4n/3n = 1.33$$

$$7.004$$

$$\text{Gflops} = 21 \text{ GB/s}$$

$$r_1 = r_0 - \alpha_1 q_1$$

$$\text{normr} = \|r_1\|$$

check convergence

for i=2,...

$$z_i = r_{i-1} M^{-1}$$

$$\rho_i = \langle r_{i-1}, z_i \rangle$$

$$\beta = \rho_i / \rho_{i-1}$$

$$3n/3n = 1.00$$

$$5.250$$

$$\text{Gflops} = 21 \text{ GB/s}$$

$$p_i = z_i + \beta p_i$$

$$2n/3n = 0.66$$

$$3.503$$

$$\text{Gflops} = 21 \text{ GB/s}$$

$$q_i = Ap_i$$

$$\alpha_i = \rho_i / \langle p_i, q_i \rangle$$

$$x_1 = x_0 + \alpha_1 z_1$$

$$\text{normx} = \|x_1\|$$

$$r_1 = r_0 - \alpha_1 q_1$$

$$\text{normr} = \|r_1\|$$

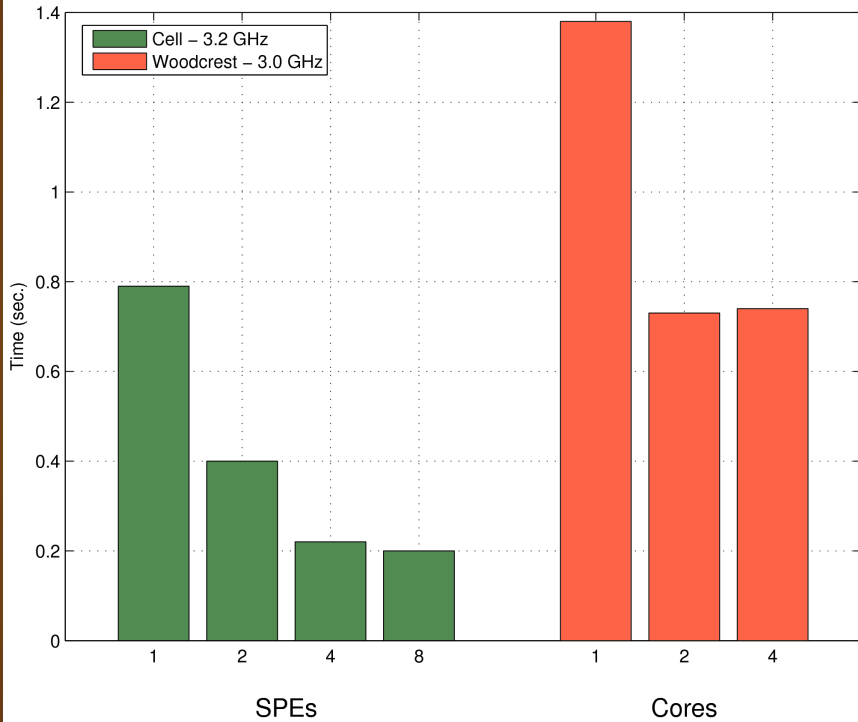
check convergence

end

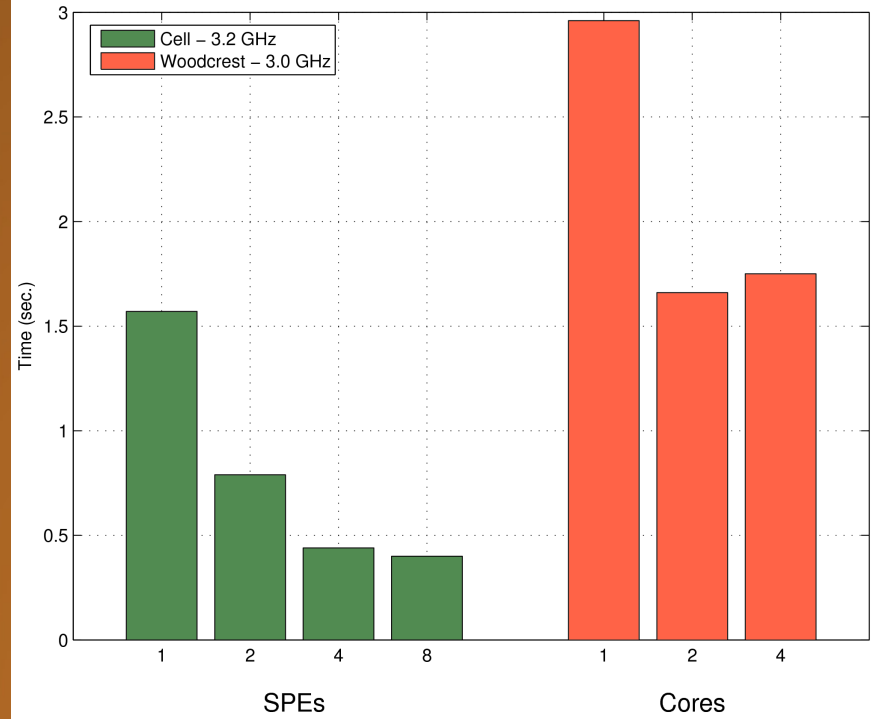
Total 6.6 Gflop/s

Sparse LA on the Cell

Problem size:819200 # iterations:80



Problem size:1638400 # iterations:80



Code on the Woodcrest is blocked, unrolled, vectorized and OpenMP parallelized. No prefetch yet.

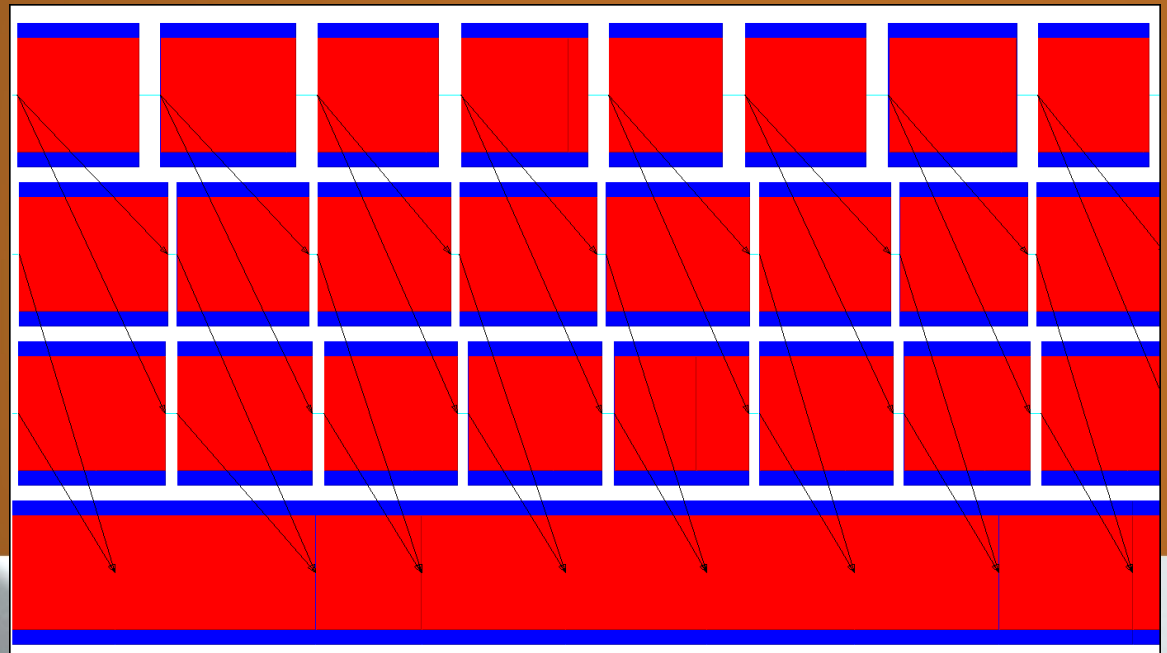
SUMMA on a 2x2 PlayStation3 cluster

What's good

- Very cheap: ~4\$ per Gflop/s (with theoretical peak)
- Fast local computations thanks to the Cell processor
- Perfect overlap between communications and computations:
 - PPE does MPI
 - SPEs do local SGEMMs

What's bad

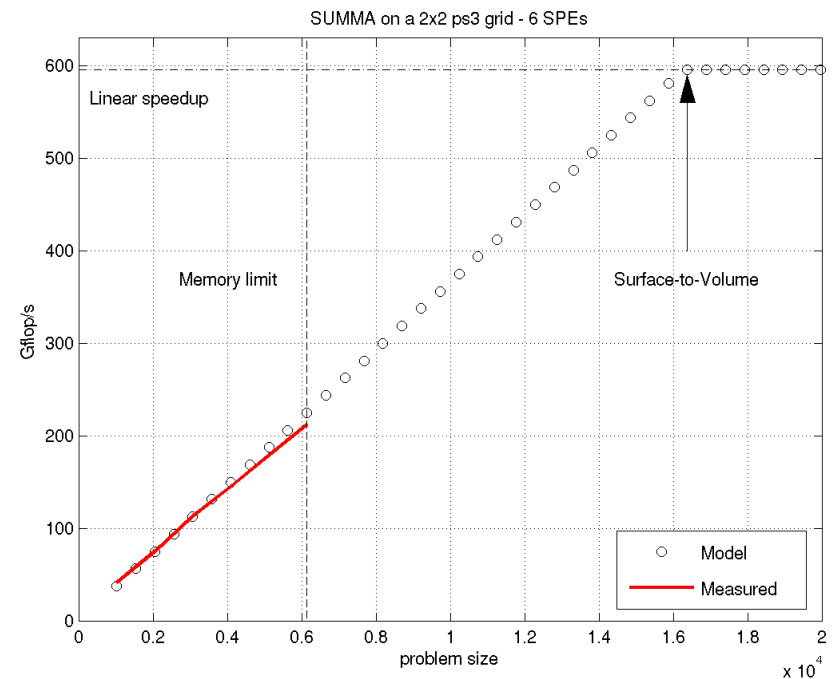
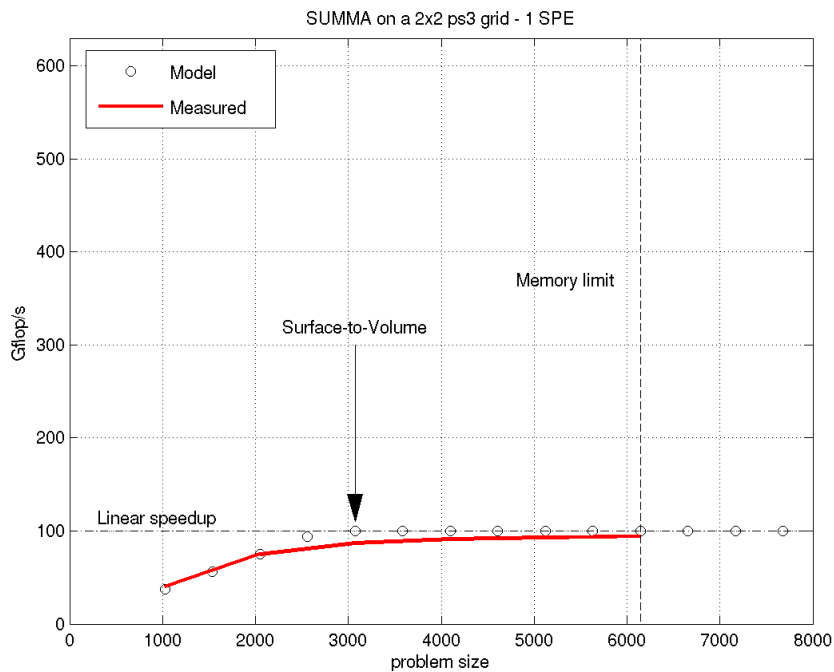
- Gigabit network card. 1 Gb/s is too little for such computational power (150 Gflop/s per node)
- Linux can only run on top of GameOS (hypervisor)
 - Extremely high network access latencies (146 usec)
 - Low bandwidth (600 Mb/s)
- Two of 8 SPEs are disabled
- Only 256 MB local memory



SUMMA on a 2x2 PlayStation3 cluster

With 6 SPEs the system runs out of memory for sizes that are too small to see the surface-to-volume effect.

Using **1 SPE** or in **Double Precision** peak performance is achieved within the system memory limits.



Thank you

<http://icl.cs.utk.edu>

<http://icl.cs.utk.edu/iter-ref>

