

Programming the Cell BE Processor

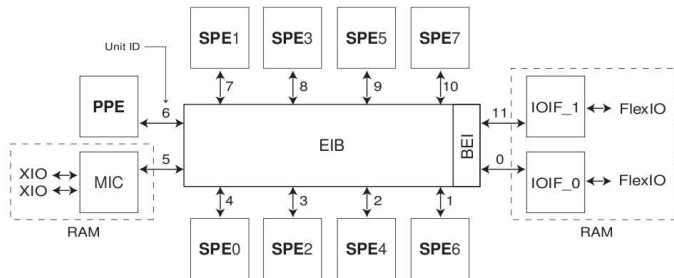
Andrea Nobile

Regensburg University

OUTLINE

- 1 THE CELL PROCESSOR
- 2 SPU ARCHITECTURE
- 3 DEPENDENCIES: PARALLELIZATION, MEMORY HIERARCHY
- 4 LQCD APPLICATION CODE: DIRAC OPERATOR
- 5 DOMAIN DECOMPOSITION
- 6 CONCLUSIONS

CELL ARCHITECTURE



BEI Cell Broadband Engine Interface
 EIB Element Interconnect Bus
 FlexIO Rambus FlexIO Bus
 IOIF I/O Interface

MIC Memory Interface Controller
 PPE PowerPC Processor Element
 RAM Resource Allocation Management
 SPE Synergistic Processor Element
 XIO Rambus XDR I/O (XIO) cell

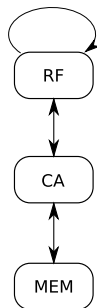
THE CELL PROCESSOR

- 9 cores (1 standard PowerPC, 8 SPU)
- More than 200 GFlops (SP) peak (mad instruction)
- High speed internal ring bus
- **Manual** main memory access from SPUs (DMA)
- Small local memory (LS), 256KB per SPU
- SIMD only, 128 128-bit registers (SPU)
- Result: Complete control and **pain** to program

HARDWARE MODEL

Simple and general hardware model:

- control
- storage (σ)
 - ▶ memory
 - ▶ caches
 - ▶ registers
 - ▶ buffers, flip-flops, fifos
- processing/transport (β, λ)
 - ▶ arithmetic pipelines
 - ▶ combinatorial logic
 - ▶ buses
 - ▶ networks



ACCESSING MAIN MEMORY FROM THE SPU

Access is done *manually*:

In order to use data stored in Main Memory with an SPU one has first to communicate a pointer to the SPU:

- PPU Code: `spe_in_mbox_write(data[num].spe_ctx,(uint32_t)&ea,1, SPE_MBOX_ALL_BLOCKING);`
- SPU Code: `ea=spu_read_in_mbox();`
- SPU Code: `spu_mfcdma(lsa, ea, size, tag, MFC_GET_CMD);`
- SPU Code: `waitag(tag_id);`

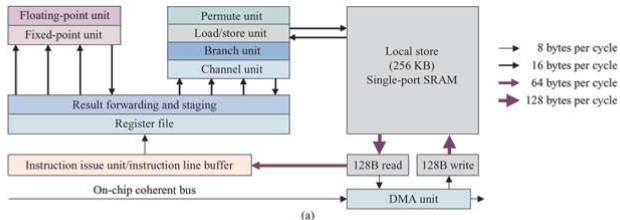
TRICKY

Accessing Main Memory on the Cell BE is like doing *asynchronous* FILE I/O on a standard processor

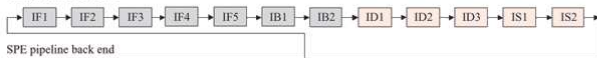
SOME IMPORTANT RESULTS FROM MICROBENCHMARKS

- Data travels in the Cell in 128B packets
- Bandwidth drops if data is moved with burst of length $< 128B$
- 256B burst length sufficient to saturate MM bandwidth
- Consequence : Try to organize data in such a way that it's possible to access it with at least 128B bursts
- Consequence : Burst must be aligned

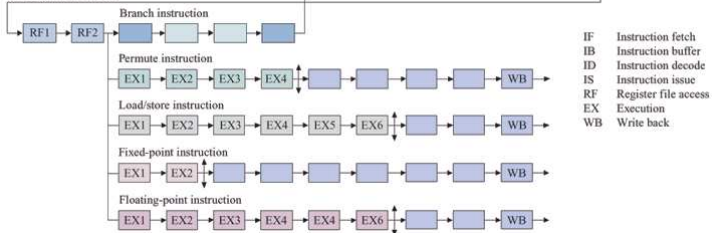
SPU ARCHITECTURE



SPE pipeline front end



SPE pipeline back end

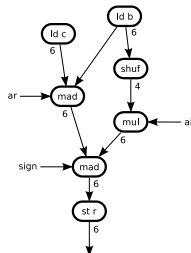


SPU SIMD PROGRAMMING EXAMPLE: COMPLEX ALGEBRA, CAXPY

- NO dedicated hardware for complex algebra
- Need to manually shuffle operands for 2- and 4-way SIMD
- **1 load/store or rotate/shuffle per cycle**
- Two layouts
 - ▶ *RIRI* (less efficient for most operations)
 - ▶ *4R4I* (needs more registers)
- Solution: **USE BOTH** depending on case, eventually change on the fly

$$r = a \times b + c$$

- *RIRI* layout
- *sign mask* (-, +, -, +)
- *a* prepared and reused



- 46% in LS (50% model), 3.2% with data in MM (4.2% model)

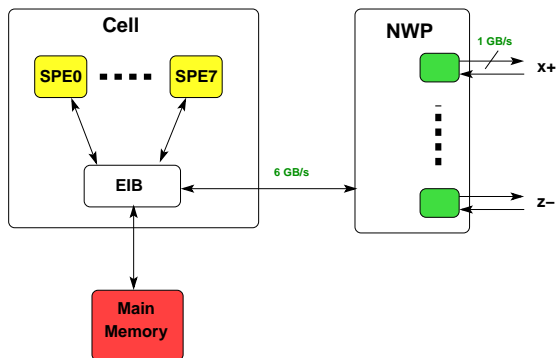
DEPENDENCY OF COMMUNICATIONS

- Non embarrassingly parallel problems require communication between computational units
- Communication between (independent) entities require a synchronization mechanism to guarantee data integrity

RAW AND WAR DEPENDENCIES

- Sender (data producer) must read the data he has produced in order to deliver them to the receiver. **RAW**
- Sender must wait the completion of the send operation before reusing the send buffer otherwise will corrupt (overwrite) the data while is being sent. **WAR**
- Receiver (data consumer) must be ready to receive data in the receive buffer when the data starts to arrive **WAR, CREDIT**
- Receiver must be notified of the completion of the data transfer before using the received data. **RAW, NOTIFY**

COMMUNICATIONS



LS-LS:

- Done with DMAs like accessing MM (LS are memory mapped)
- Communication dependencies must be enforced with a software mechanism
- Relatively easy to manage (fast EIB)

COMMUNICATIONS

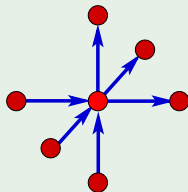
Communications via torus network

- 6 GByte/s aggregate throughput
- 1GByte/s per link per direction
- Hardware **credit** and **notify** mechanism
- Lightweight and reliable protocol
- 128B packets
- Low latency LS - remote LS communication through an hardware *channel* mechanism
- Identity of messages enforced by order (no MPI-like tags)

TRADEOFF: STORAGE VS INFORMATION EXCHANGE

TWO EXTREME CASES

- In the case of a big enough LS to hold the whole Cell local lattice, plus the storage required for the chosen schedule, we need only to load the input data one time (also if the data is accessed many times), do the computation, and then store back the result in main memory.
- In the case of an extremely small LS then we are forced to move back and forth input data and partial results (details depending on the schedule) without any possibility to *reuse* data already loaded.



TRADEOFF: STORAGE VS LATENCY

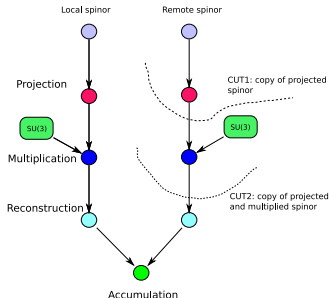
- Given sufficient storage it is possible to hide latency of communications by overlapping communications with computations and doing adequate prefetch.
- If available bandwidth is not sufficient, complete overlap is not possible
- The more the latency you want to be able to tolerate the more is the storage requirement. Prefetch must be done more time ahead and you need to have sufficient storage to hold data for all the *in-flight* fetches.
- Concept clearly valid both for memory accesses and communication since the only difference is the data path.

DIRAC OPERATOR: PARALLELIZABILITY

We consider the hopping term D_h of the Dirac operator

$$D_w + m = \frac{1}{2\kappa}(1 - \kappa D_h) \quad (1)$$

$$D_h\phi(x) = \sum_{\mu=0}^3 \left[U(x, \mu)(1 - \gamma_\mu)\phi(x + \hat{\mu}) + U^\dagger(x - \hat{\mu}, \mu)(1 + \gamma_\mu)\phi(x - \hat{\mu}) \right] \quad (2)$$



- divide lattice among processors
- storage and information exchanges dependent on schedule and allocation (NP hard)
- need to communicate data
- low level parallelism exploited through SIMD

DIRAC OPERATOR: SLICE MULTIBUFFERING

3D slice multibuffering technique

CHALLENGES

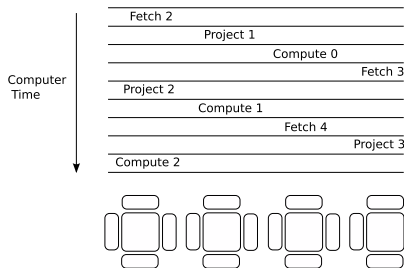
- Not straightforward to code
- Not straightforward synchronization (no barrier)

ADVANTAGES

- Optimal data reuse
- Tolerate moderate network latencies
- **Memory bandwidth limited on the Cell**
- Need to optimize memory accesses (minimize I_{LM})
- No SU(3) sharing

DIRAC OPERATOR: HOW IT WORKS

- local lattice is tiled with 3d-slices that fit in LS
- “computation” slice is moved through the lattice along the t direction



$$S = L^3(3 \cdot 4|U| + 4|\phi|) + (4 \cdot 6L^2/2 + 2 \cdot 6L^2/2 + 2L^2)|\phi| \quad (3)$$

PERFORMANCE ESTIMATE

- FP efficiency limited to 34% by MM bandwidth (27% corrected)
- Not straightforward to reach 34% performance (complex code and communication commands)

DIRAC OPERATOR: DATA LAYOUT

MACROSCOPIC LAYOUT

- For each SPU a set of buffers are allocated for spinor and gauge fields
- Cell local lattice in this way is already sectioned into SPU-friendly buffers

MICROSCOPIC LAYOUT

- *RIRI* layout (faster projections)
- Standard layout for SU(3) matrices but padded to 80 B (5 registers)
- Unusual spinor layout
 - ▶ We want *monochromatic* registers
 - ▶ We want to access separately Dirac indexes 0-1 and 2-3
 - ▶ Constraints are satisfied with $[\psi_0^0, \psi_1^0, \psi_0^1, \psi_1^1, \dots]$
- SU(3)-spinor multiplication is performed with *4R4I* layout
- Handling of boundaries without any *if* statement

DIRAC OPERATOR: COMMUNICATIONS AND SYNCRONIZATION

- Only projected spinors are communicated (one slice ahead)
- Data integrity guranteed with two mechanisms
 - ▶ Sender: After sending data, write a (remote) location with notify code. Receiver: Before using arrived data, clear the notify location.
 - ▶ Sender: Prior to send data, check if (remote) buffers are free by testing the (local) “check” location. After sending, mark the “check” location as not free. Receiver: After using data, notify neighbors (write to “check” location) that the buffers are free.
- Tolerate a moderate asynchronous execution
- No barrier

DIRAC OPERATOR: BENCHMARKS

L	4	5
$L_0 = 32$	25%	24%
$L_0 = 64$	26%	25%
$L_0 = 128$	26%	26%

- As estimated with the performance model with MM efficiency correction
- Increasing L_0 performance increases
- QS20, 64KB pages

DIRAC OPERATOR: CONSIDERATIONS

- Limited flexibility (spatial lattice sizes)
- Linear algebra likely to degrade performance with high number of Krylov vectors

REALISTIC SIMULATIONS

- Need a more complex operator (e/o + $O(a)$ improvement), performance will be limited to **16%**
- Increasing flexibility decreases performance and increases code complexity

DOMAIN DECOMPOSITION TECHNIQUES

- Reduce MM Information exchange
- Increase flexibility
- **Complex algorithm and code**

DOMAIN DECOMPOSITION AS A PRECONDITIONER

The SAP procedure consists in a solver. The domain on which the equation is to be solved is divided into (sub)-domains (blocks), which are chess-board coloured, with Dirichlet boundary conditions. The core of the procedure is the **block solver**.

- The SAP procedure converges slowly
- Luescher's idea: Use it as a preconditioner

The righ-preconditioned Dirac equation reads

$$DM_{sap}\phi = \rho \quad (4)$$

once solved, the original solution $\psi = D^{-1}\rho$ is obtained through

$$\psi = M_{sap}\phi \quad (5)$$

- Uses a GMRESR Krylov space algorithm
- Mixed precision (1% double)

DD: BLOCK SOLVER

- Single precision
- Uses a simple MR algorithm
- **Physical relevant block sizes fit LS**
- Possibility to “overcome” MM bandwidth limit if well implemented
- e/o + clover

DD: PARALLEL BLOCK SOLVER

A single SPU can hold a 4^4 block. In order to simulate with bigger blocks it is necessary to parallelize the block solver.

- Block is divided along the t direction on the 8 SPUs
- Up to a $8 \cdot 6 \cdot 6 \cdot 8$ block
- Double buffering scheme for communications
- Communicate unprojected spinors (fast EIB)
- Threads mapped on bus ring topology
- Complete communication-computation overlap (\hat{Q})
- simple synchronization mechanism

DD: BLOCK SOLVER PERFORMANCE

The benchmarks of the block solver are satisfactory

MEASURED PERFORMANCE

- Simple block solver achieves **54%** FP efficiency
- Parallel block solver can sustain **50%** of the peak FP performance
- Timings do not take into account the time spent in loading spinor and clover fields into local store.
- We expect $\approx 30\%$ performance for the complete solver.

CONCLUSIONS

POWERFUL BUT TRICKY!

- The Cell BE architecture is powerful but poses significant challenges to the programmer.
- Main challenges and potential from architectural features
 - ▶ unbalance between MM bandwidth and FP performance
 - ▶ small LS
 - ▶ manual MM access
- The concept of *porting* software engineered for a classic processor on the Cell processor doesn't generally apply, **rethinking and rewriting** sounds better.
- All this stuff and more: in preparation with H. Simma.

DIRAC OPERATOR: INDEXING

PROBLEMS

- Unbalance between FP SIMD power and scalar integer (addresses) calculations
- Poor address calculation and load/store instruction selection by compiler
- Boundaries need special treatment

SOLUTION

- Precalculated *offsets*
- Pointers obtained on the fly with SIMD code
- Explicit placement of needed pointer into preferred slot (rotqwb)

HANDLING OF BOUNDARIES

- No *if* statements
- Fixed code, uses neutral elements through calculated pointers

DIRAC OPERATOR: DETAILS

- Offset tables calculated on the PPU at initialization, each offset is a short unsigned int
- Tables contains offsets (respect to the slice structure address) of spinors and SU(3) matrices that are neighbours to a given spinor.
- If a neighbour belong to the external border its offset can point to a neutral element (zero weyl spinor/identity SU(3))
- Offset tables are transferred to the SPU with a DMA
- Pointers are obtained on the fly by expanding the unsigned short ints into unsigned ints by shuffle instructions then adding with a SIMD add instruction the 32bit offsets with a *splatted* pointer (4 pointers are obtained in a single cycle instead of several cycles for a single pointer)
- A single offset table is needed for all the SPU slices.
- Code is free of *if* statements inside the most critical section.

DD: O(A) IMPROVED, E/O PRECONDITIONED DIRAC OPERATOR

The Dirac operator used is the O(a) improved (clover) Wilson-Dirac operator. The block solver is preconditioned with even-odd preconditioning.

$$D = D_w + m + c_{sw} \sum_{\mu, \nu=0}^3 \frac{i}{4} \sigma_{\mu\nu} \hat{F}_{\mu\nu} \quad (6)$$

The form used is

$$Q = \gamma_5 D \quad (7)$$

The even-odd preconditioned Dirac operator is defined as

$$\hat{Q} = Q_{ee} - Q_{eo} Q_{oo}^{-1} Q_{oe} \quad (8)$$

The Dirac equation $Q\psi = \eta$ is solved by first solving

$$\hat{Q}\psi_e = \eta_e - Q_{eo} Q_{oo}^{-1} \eta_o \quad (9)$$

The odd field is obtained with

$$\psi_o = Q_{oo}^{-1} \{ \eta_o - Q_{oe} \psi_e \} \quad (10)$$

DD: DATA ALLOCATION SCHEME

Maximum efficiency and flexibility are obtained if

- The 3 needed spinor fields are kept in LS (MR)
- The SW (clover) field is kept in LS
- The gauge field is loaded from MM during any application of \hat{Q}
- Gauge field reload proceed with plane granularity