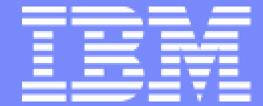


Cell Broadband Engine

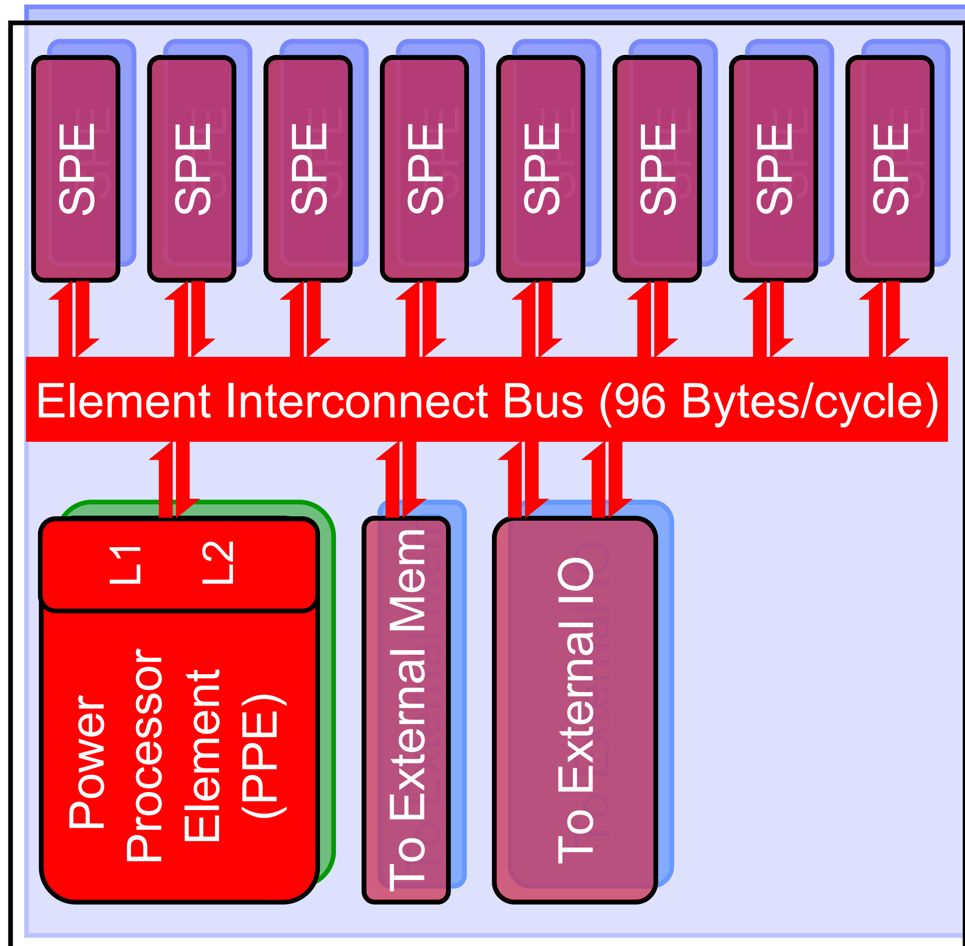
Optimizing Compiler for the Cell BE

Kathryn O'Brien
Kevin O'Brien
Alexandre Eichenberger
Peng Wu

Presented by Roch Archambault, archie@ca.ibm.com

Ryerson University, October 26, 2006

Cell Broadband Engine Overview



8 Bytes (per dir)

16Bytes (one dir)

128Bytes (one dir)

- **Heterogeneous, multi-core engine**
  - 1 multi-threaded power processor
  - up to 8 compute-intensive-ISA engines
- **Local Memories**
  - fast access to 256KB local memories
  - globally coherent DMA to transfer data
- **Pervasive SIMD**
  - PPE has VMX
  - SPEs are SIMD-only engines
- **High bandwidth**
  - fast internal bus (200GB/s)
  - dual XDR™ controller (25.6GB/s)
  - two configurable interfaces (76.8GB/s)
  - numbers based on 3.2GHz clock rate

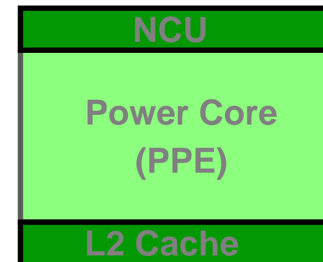
© 2006 IBM Corporation

Cell Processor Components

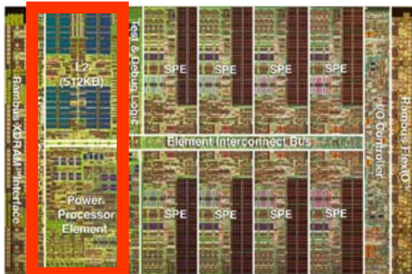
Power Processor Element (PPE)

- ❑Industry-standard 64-bit IBM Power Architecture™ processor
  - ➢ PowerPC AS 2.0.2
- ❑2-Way Hardware Multithreaded
- ❑L1 : 32KB I ; 32KB D
- ❑L2 : 512KB
- ❑Coherent load/store
- ❑VMX
- ❑3.2+ GHz
- ❑Realtime Control
  - ➢ Locking L2 Cache & TLB
  - ➢ Bandwidth Reservation

In the Beginning
– the Power Architecture™ Processor



Custom Designed
– for high frequency, area and power efficiency

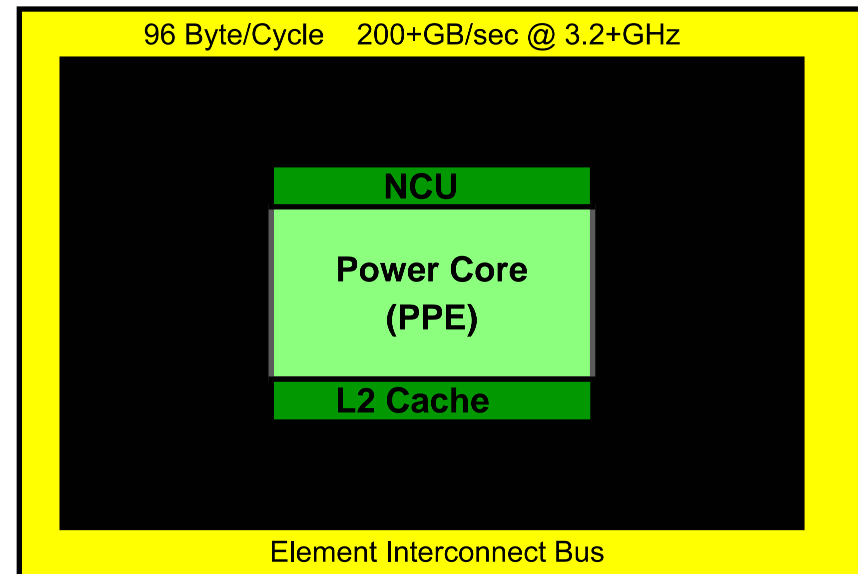


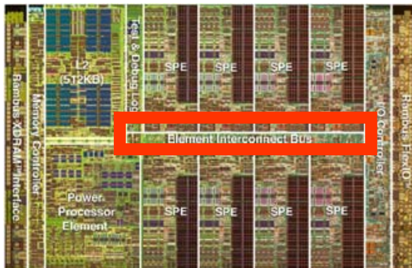
© 2006 IBM Corporation

Cell Processor Components

Element Interconnect Bus

- data ring for internal communication
  - Four 16 byte data rings, supporting multiple transfers
  - 96B/cycle peak bandwidth
  - Over 100 outstanding requests
  - 200+ GByte/s @ 3.2+ GHz



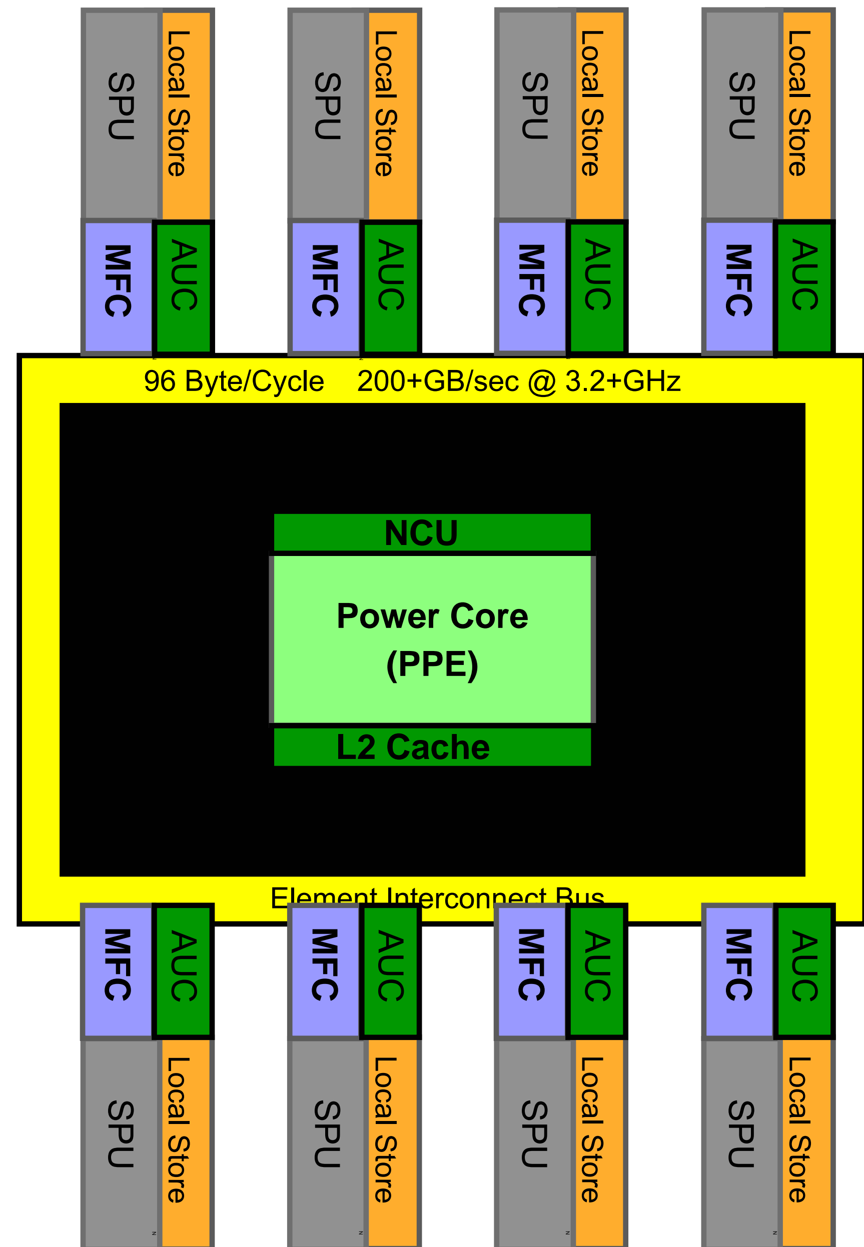


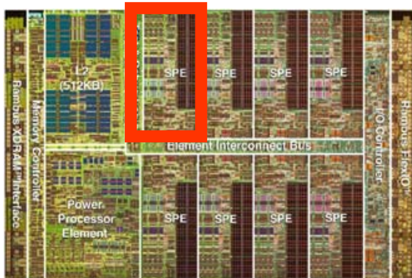
© 2006 IBM Corporation

Cell Processor Components

SPE provides computational performance

- ❑Dual issue, up to 8-way 32-bit SIMD
- ❑Dedicated resources
  - ➢ 128-entry 128-bit VRF
  - ➢ 256KB Local Store
- ❑Each SMF can be dynamically configured to protect resources
- ❑Dedicated DMA engine
  - ➢ Up to 16 outstanding requests



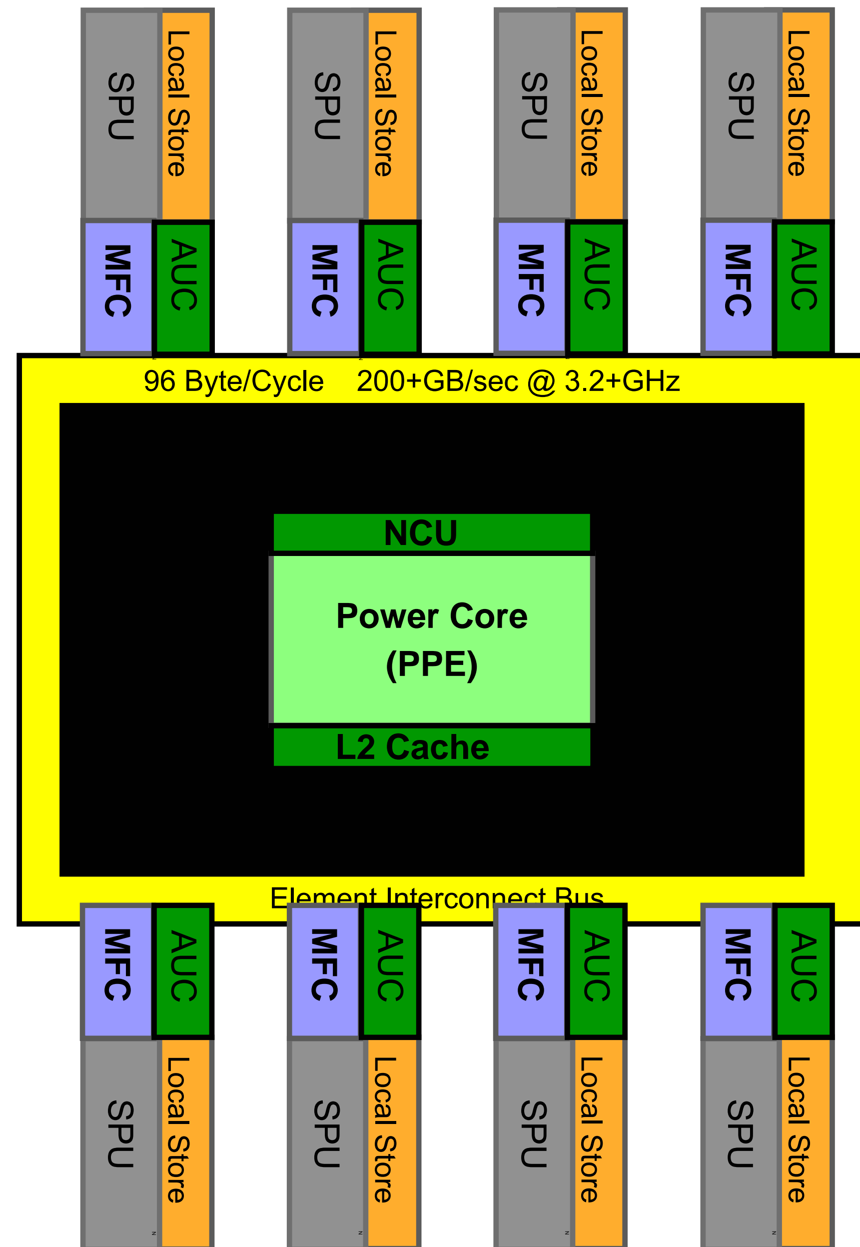


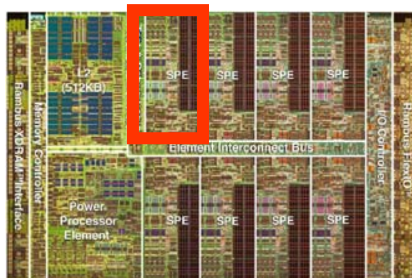
© 2006 IBM Corporation

Cell Processor Components

SMF provides memory management & mapping

- ❑SPE Local Store aliased into system memory map
- ❑SMF controls SPE DMA accesses
  - ➢ Compatible with Power Architecture™ Virtual Memory architecture
  - ➢ S/W controllable from PPE MMIO
- ❑DMA 1,2,4,8,16,128 B ⇒ 16Kbyte transfers for I/O access
- ❑SPE DMA access protected by SMF
  - ➢ Based on Power Architecture™ system memory map





© 2006 IBM Corporation

Cell Processor Components

I/O provides wide bandwidth

- Dual XDR™ controller
  - 25.6GB/s @ 3.2Gbps
- Two configurable interfaces
  - 76.8GB/s @ 6.4Gbps
  - Configurable number of Bytes
  - Coherent or I/O Mode Interconnect
- Supports multiple system configurations

20 GB / s
BIF or IOIF1

SPU | Local Store | MFC | AUC (×8)

IOIF1
IOIF0

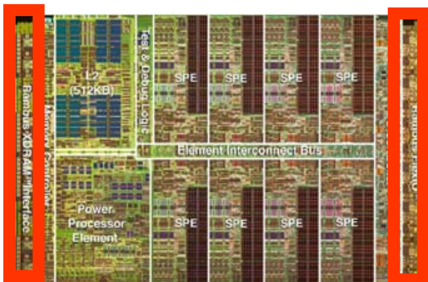
96 Byte/Cycle 200+GB/sec @ 3.2+GHz

NCU
Power Core
(PPE)
L2 Cache

Element Interconnect Bus

MIC
25 GB / s
DRAM

5 GB / s
Southbridge
I/O



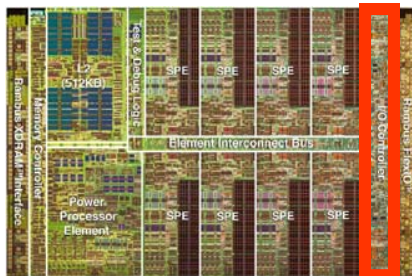
© 2006 IBM Corporation

Cell Processor Components

IIC – Internal Interrupt Controller

- ❑Handles SPE Interrupts
- ❑Handles External Interrupts
  - ➢ From Coherent Interconnect
  - ➢ From IOIF0 or IOIF1
- ❑Interrupt Priority Level Control
- ❑Interrupt Generation ports for IPI
- ❑Duplicated for each PPE hardware thread

20 GB / s
BIF or IOIF1
IOIF1
SPU
Local Store
MFC
AUC
96 Byte/Cycle 200+GB/sec @ 3.2+GHz
NCU
Power Core
(PPE)
L2 Cache
IIC
MIC
25 GB / s
DRAM
IOIF0
Element Interconnect Bus
5 GB / s
Southbridge
I/O



© 2006 IBM Corporation

Cell Processor Components

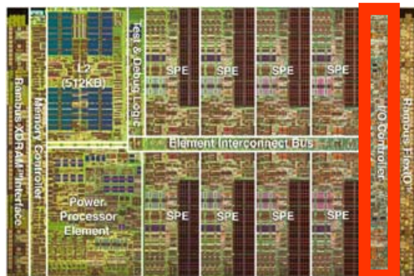
IOT implements I/O Bus Master Translation

- ❑Translates bus address to system address
- ❑Two Level translation
  - ➢ I/O Segments: 256 MB
  - ➢ I/O Pages: 4KB, 64KB,1MB, 16MB
- ❑I/O Device Identifier / page for LPAR
- ❑IOST and IOPT Cache
  - ➢ hardware/software managed

20 GB / s
BIF or IOIF1

SPU | Local Store | MFC | AUC (×8)

IOIF1
96 Byte/Cycle 200+GB/sec @ 3.2+GHz
NCU
Power Core
(PPE)
L2 Cache
IIC
IOT
Element Interconnect Bus
MIC
25 GB / s
DRAM
IOIF0
5 GB / s
Southbridge
I/O



© 2006 IBM Corporation

Cell Processor Components

Token Manager provides Bandwidth Reservation for shared resources

- Optionally used for RT tasks or LPAR
- Multiple Resource Allocation Groups
- Generates access tokens at configurable rate for each allocation group
  - 1 per each memory bank (16 total)
  - 2 for each IOIF (4 total)
- Requestors assigned RAG ID by OS/hypervisor
  - Each SPE
  - PPE L2 / NCU
  - IOIF 0 Bus Master
  - IOIF 1 Bus Master
- Priority order for using another RAGs unused tokens
- Resource overcommit warning interrupt

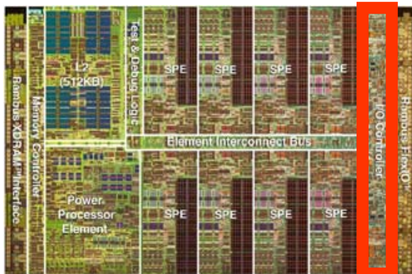
20 GB / s
BIF or IOIF1

SPU
Local Store
MFC
AUC

IOIF1
96 Byte/Cycle 200+GB/sec @ 3.2+GHz
NCU
Power Core
(PPE)
L2 Cache
IIC
TKM
IOT
Element Interconnect Bus
IOIF0

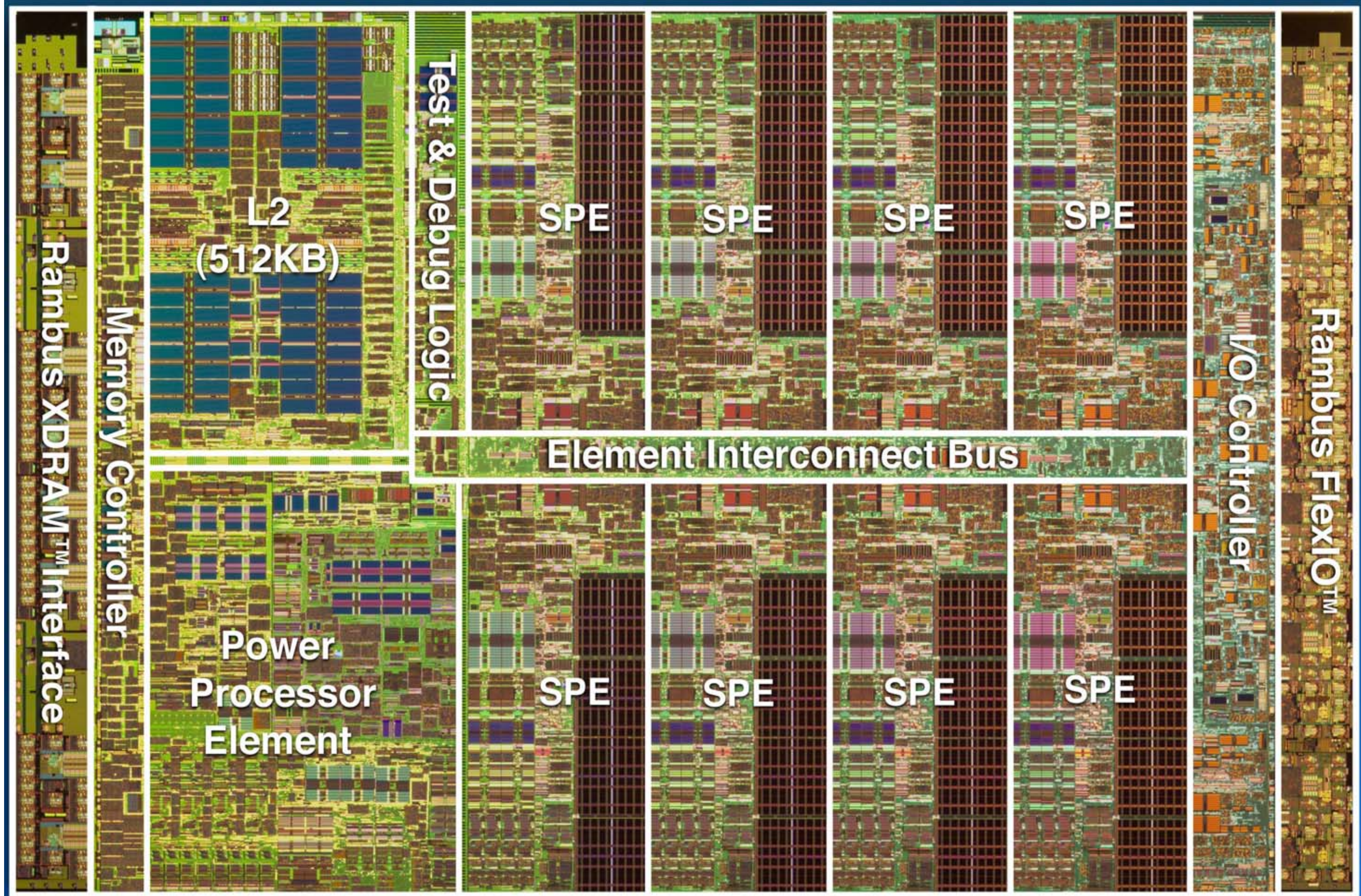
MIC
25 GB / s
DRAM

5 GB / s
Southbridge
I/O



© 2006 IBM Corporation

Cell Broadband Engine



© 2006 IBM Corporation

Supporting a Broad Range of Expertise to Program Cell

Highest performance with help from programmers

Multiple-ISA hand-tuned programs

PROGRAMS

Automatic tuning for each ISA

Explicit SIMD coding

SIMD

SIMD/alignment directives

Automatic simdization

Explicit parallelization with local memories

PARALLELIZATION

Shared memory, Single program abstraction

Automatic parallelization

Highest Productivity with fully automatic compiler technology

© 2006 IBM Corporation

Outline

Part 1:
Automatic SPE tuning

Part 2:
Automatic simdization

Part 3:
Shared memory &
Single program abstr.

Multiple-ISA hand-tuned programs

Automatic tuning for each ISA

Explicit SIMD coding

SIMD/alignment directives

Automatic simdization

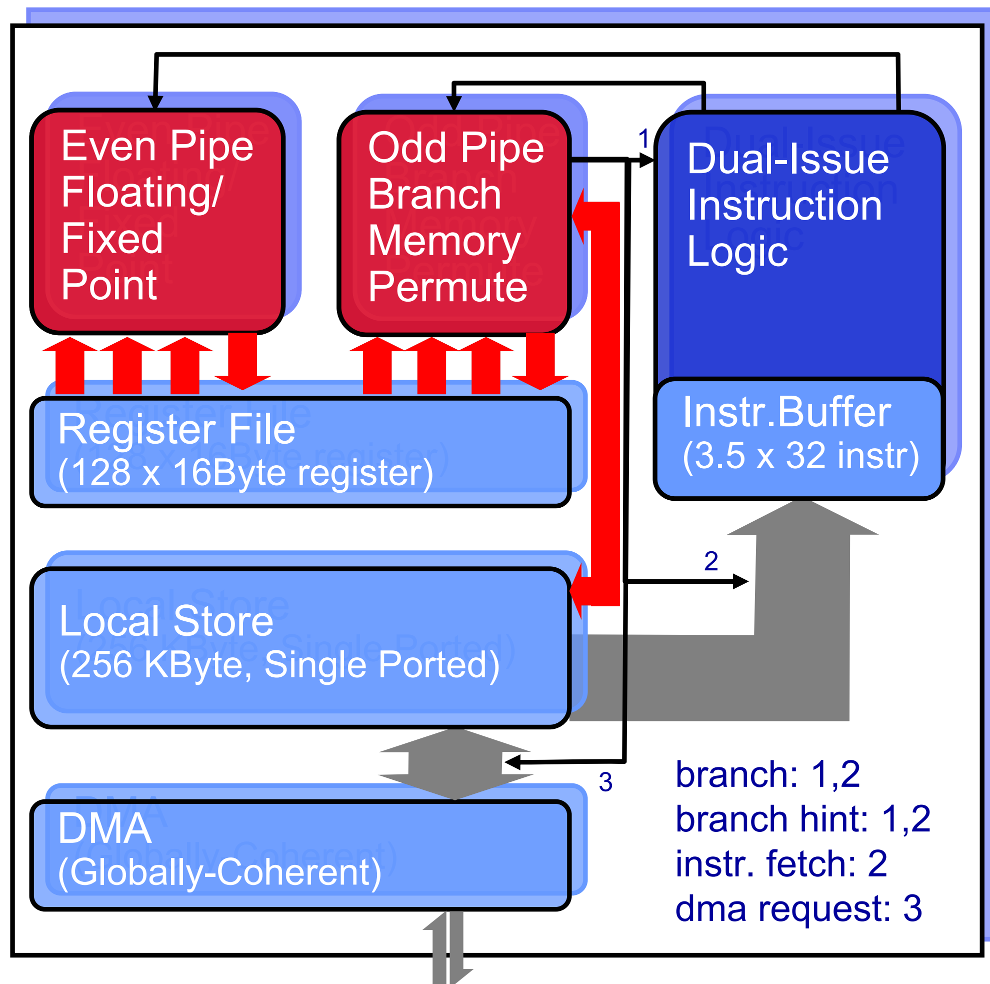
Explicit parallelization with local memories

Shared memory, single program abstraction

Automatic parallelization

© 2006 IBM Corporation

SPE's Functional Units are SIMD Only

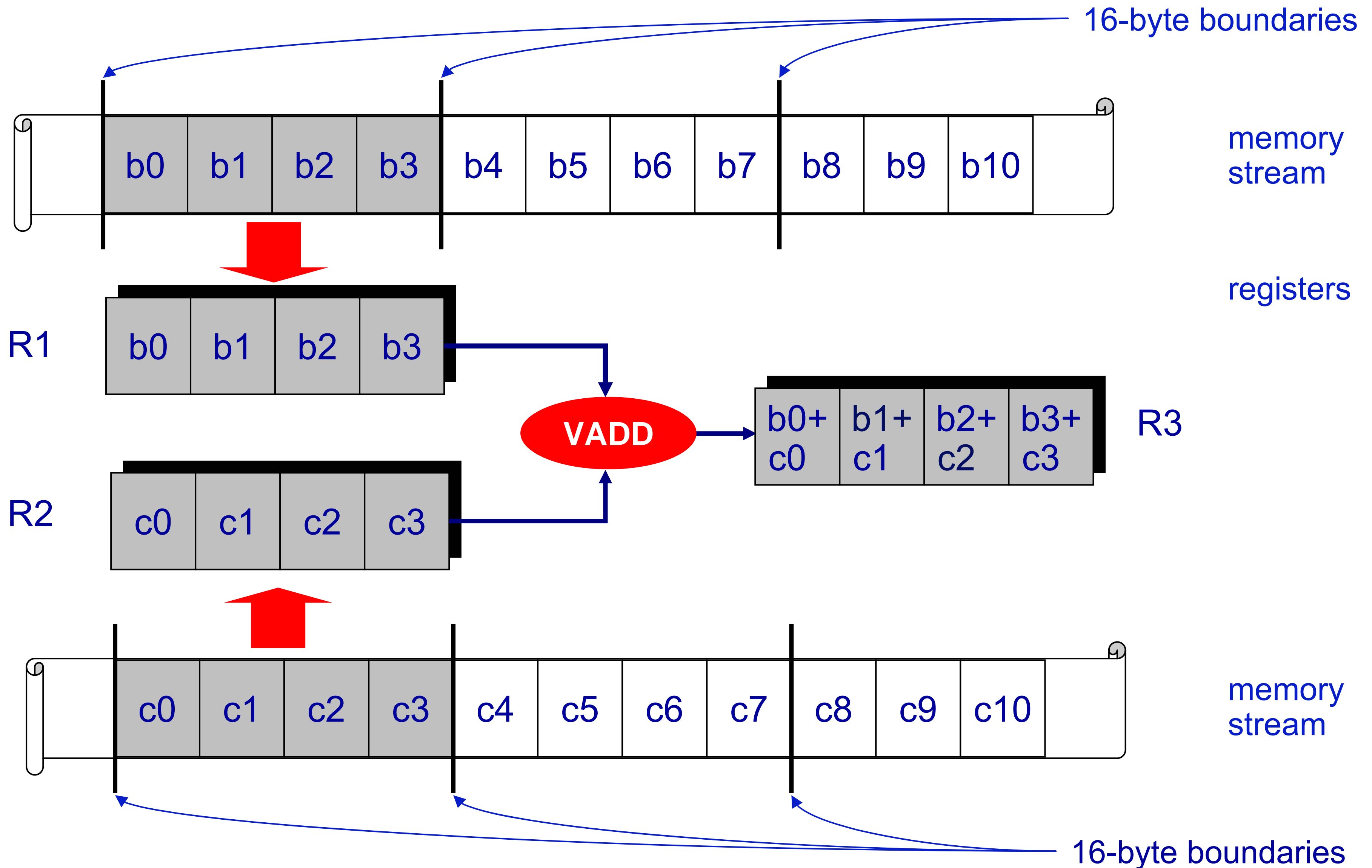


- **Functional units are SIMD only**
  - all transfers are 16 Bytes wide,
  - including register file and memory
- **How to handle scalar code?**

© 2006 IBM Corporation

Single Instruction Multiple Data (SIMD)

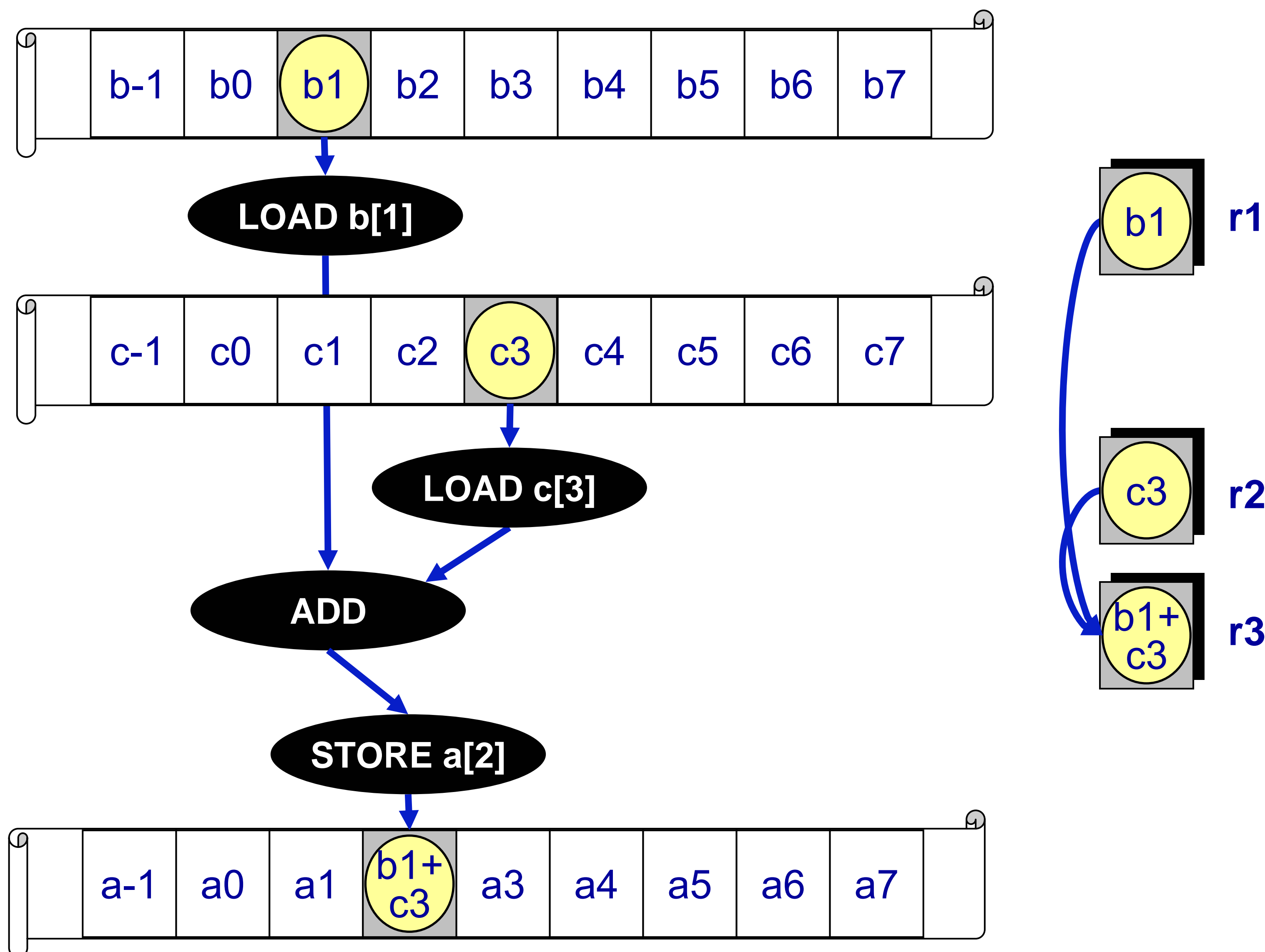
Meant to process multiple “b[i]+c[i]” data per operations



© 2006 IBM Corporation

Scalar code on Scalar Functional Units

- Example: a[2] = b[1] + c[3]



© 2006 IBM Corporation

Scalar Code on SIMD Functional Units

- **Example: a[2] = b[1] + c[3]**

b-1 | b0 | b1 | b2 | b3 | b4 | b5 | b6 | b7

LOAD b[1]

16-byte boundaries

c-1 | c0 | c1 | c2 | c3 | c4 | c5 | c6 | c7

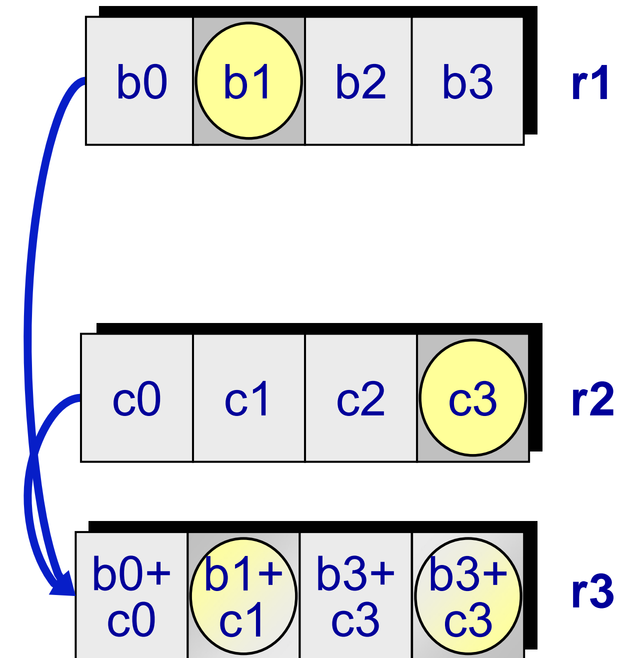
LOAD c[3]

ADD

STORE a[2]

a-1 | b0+ c0 | b1+ c1 | b2+ c2 | b3+ c3 | a4 | a5 | a6 | a7

Problem #1:
Memory alignment defines data location in register



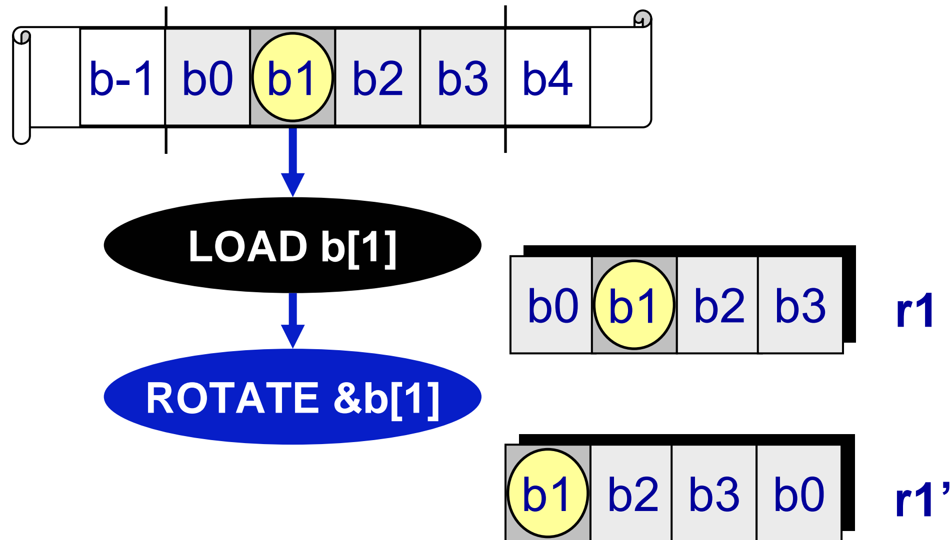
Problem #2:
Adding aligned values yield wrong result

Problem #3:
Vector store clobbers neighboring values

© 2006 IBM Corporation

Scalar Load Handling

- **Use read-rotate sequence**



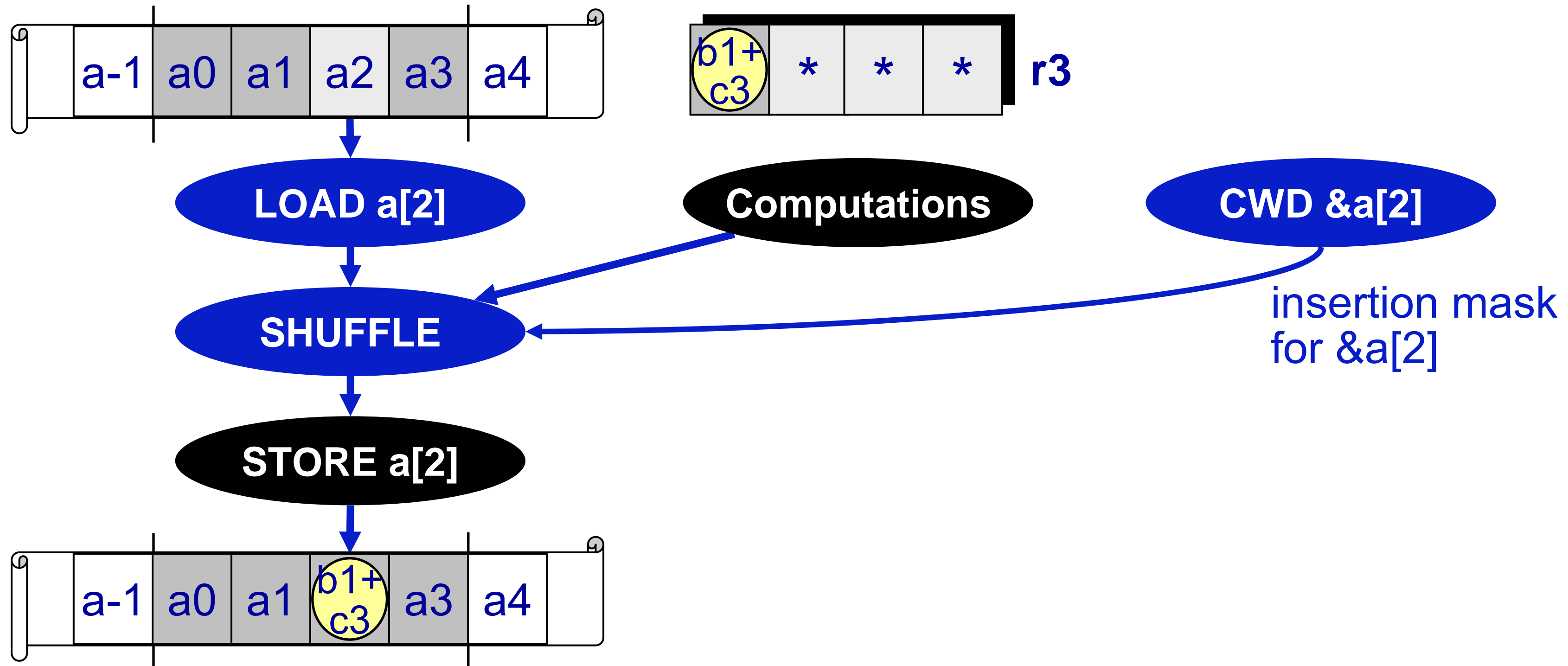
- **Overhead (1 op, in blue)**
  - one quad-word byte rotate

- **Outcome**
  - desired scalar value always in the first slot of the register
  - this addresses Problems 1 & 2

© 2006 IBM Corporation

Scalar Store Handling

- ❑ **Use read-modify-write sequence**



- ❑ **Overhead (1 to 3 ops, in blue)**
  - one shuffle byte, one mask formation (may reuse), one load (may reuse)

- ❑ **Outcome**
  - SIMD store does not clobber memory (this addresses Problem 3)

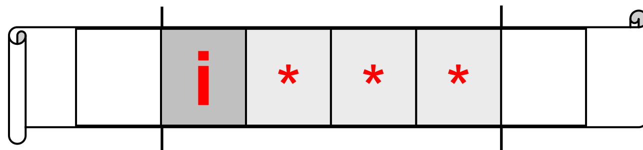
© 2006 IBM Corporation

Optimizations for Scalar on SIMD

- **Significant overhead for scalar load/store can be lowered**

- **For vectorizable code**
  - generate SIMD code directly to fully utilize SIMD units
  - done by expert programmers or compilers (see SIMD presentation)

- **For scalar variable**
  - allocate scalar variables in first slot, by themselves

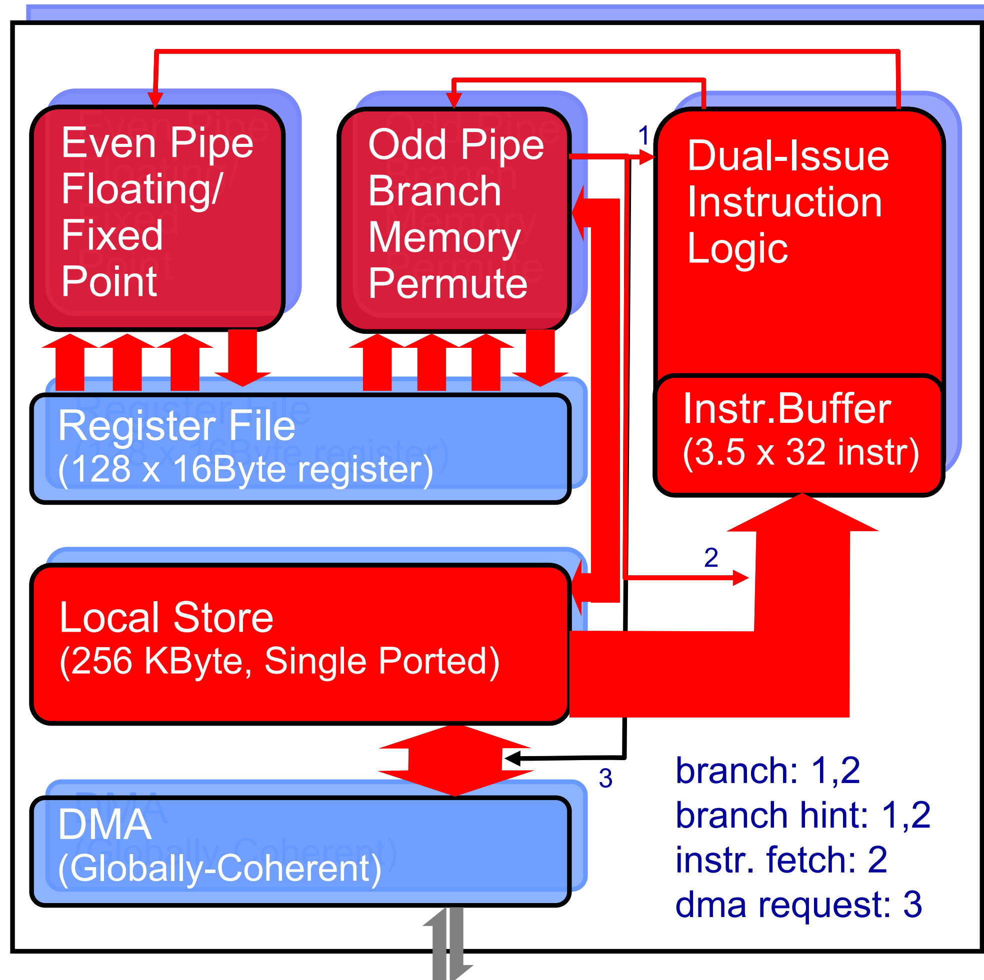


  - eliminate need for rotate when loading
    - data is guaranteed to be in first slot (Problems 1 & 2)
  - eliminate need for read-modify-write when storing
    - other data in 16-byte line is garbage (Problem 3)

© 2006 IBM Corporation

SPE Features Optimized for by the Compiler

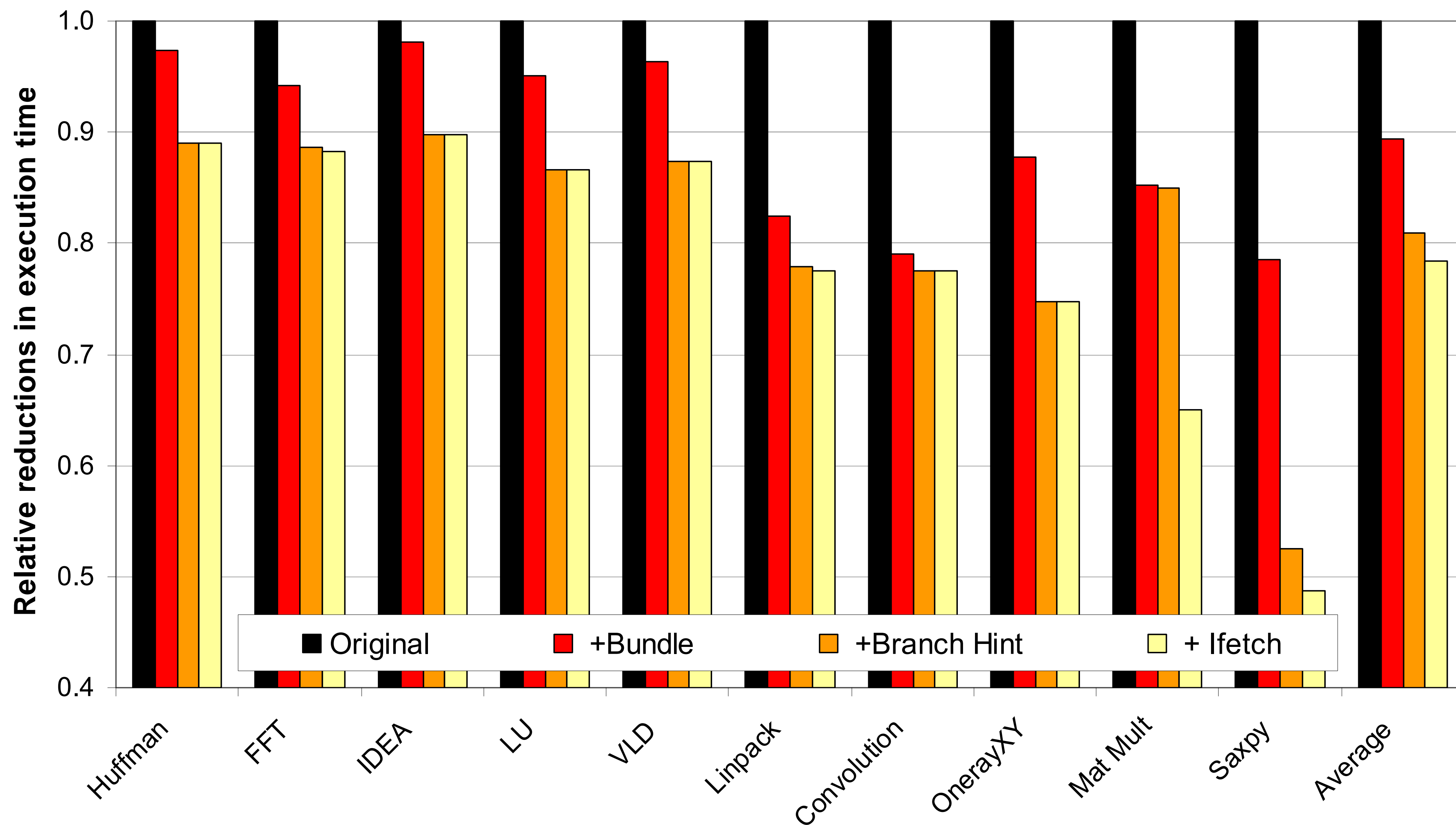
Synergistic Processing Element (SPE)



- **SIMD-only functional units**
  - 16-bytes register/memory accesses
- **Simplified branch architecture**
  - no hardware branch predictor
  - compiler managed hint/predication
- **Dual-issue for instructions**
  - full dependence check in hardware
  - must be parallel & properly aligned
- **Single-ported local memory**
  - aligned accesses only
  - contentions alleviated by compiler

© 2006 IBM Corporation

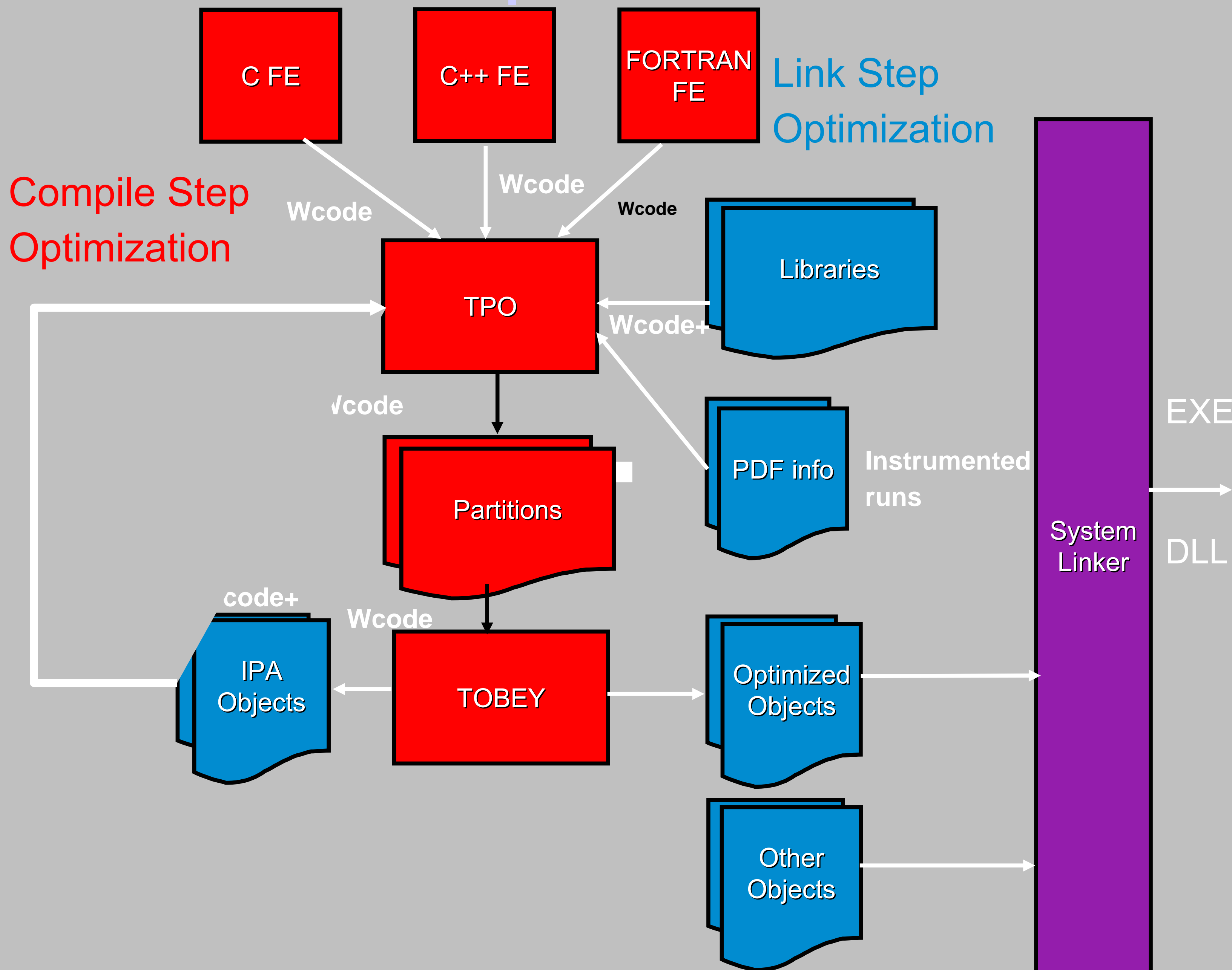
SPE Optimization Results



single SPE performance, optimized, simdized code (avg 1.00 → 0.78)

© 2006 IBM Corporation

IBM XL Compiler Architecture



© 2006 IBM Corporation

Outline

Part 1:
Automatic SPE tuning

Part 2:
Automatic simdization

Part 3:
Shared memory &
Single program abstr.

Multiple-ISA hand-tuned programs

Explicit SIMD coding

Explicit parallelization with local memories

PROGRAMS

SIMD

SIMD/alignment directives

PARALLELIZATION

Shared memory, single program abstraction

Automatic tuning for each ISA

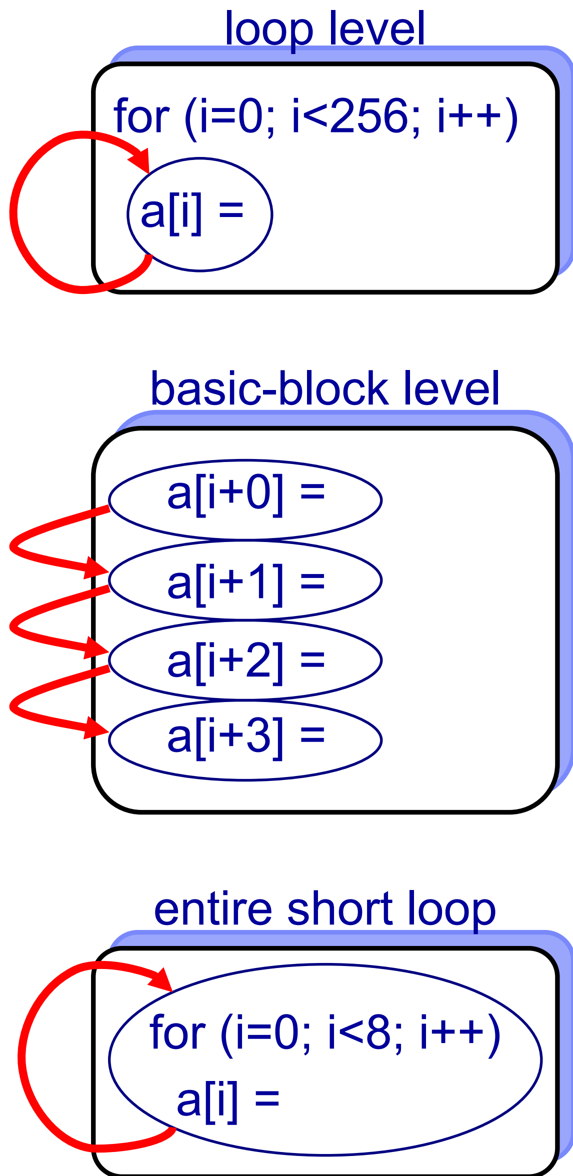
Automatic simdization

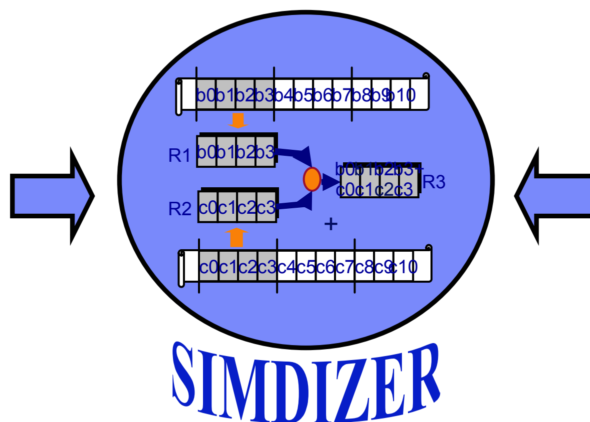
Automatic parallelization

© 2006 IBM Corporation

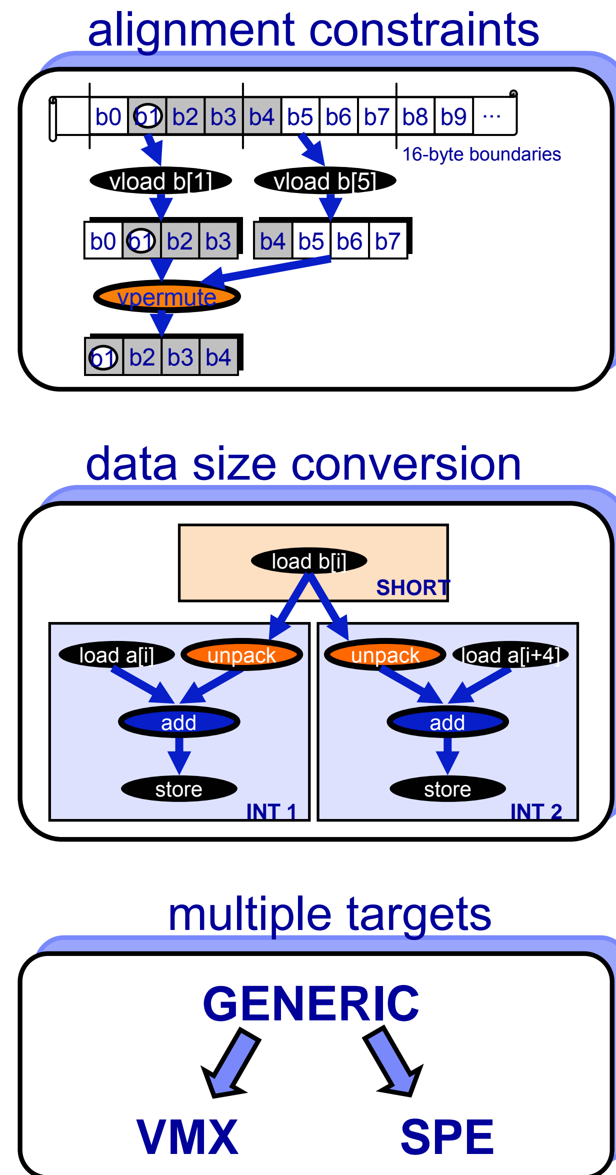
Successful Simdization

Extract Parallelism



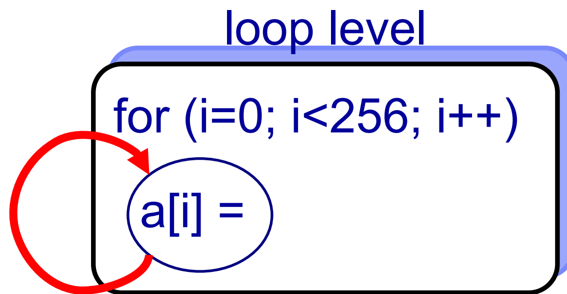


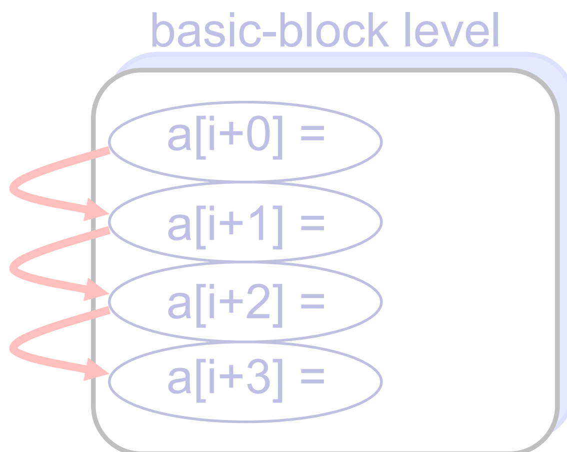
Satisfy Constraints

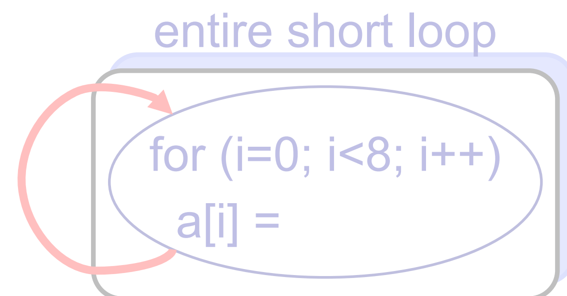


© 2006 IBM Corporation

Example of SIMD-Parallelism Extraction







Loop level

- SIMD for a single statement across consecutive iterations
- successful at:
  - efficiently handling misaligned data
  - pattern recognition (reduction, linear recursion)
  - leverage loop transformations in most compilers

[Bik *et al*, IJPP 2002]

[VAST compiler, 2004]

[Eichenberger *et al*, PLDI 2004] [Wu *et al*, CGO 2005]

[Naishlos, GCC Developer's Summit 2004]

© 2006 IBM Corporation

Example of SIMD Constraints

- **Alignment in SIMD units matters:**
  - consider “b[i+1] + c[i+0]”

16-byte boundaries

b0 b1 b2 b3 b4 b5 b6 b7

vload b[1]

R1 b0 b1 b2 b3

R2 c0 c1 c2 c3

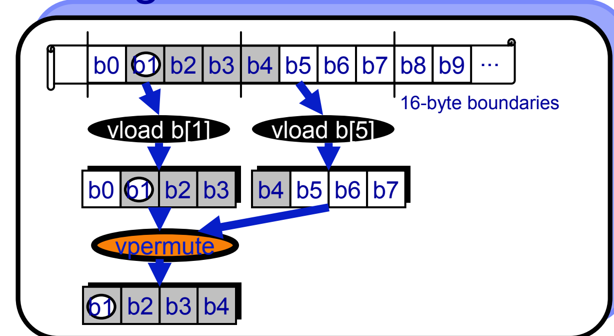
b0+c0 | b1+c1 | b2+c2 | b3+c3

this is not b[1] + c[0]

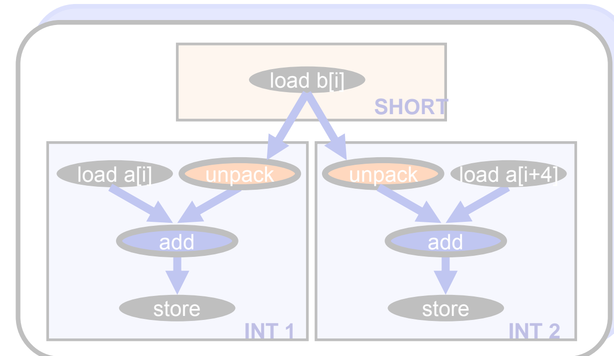
c0 c1 c2 c3 c4 c5 c6 c7

16-byte boundaries

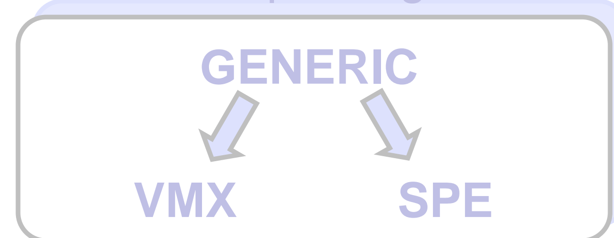
alignment constraints



data size conversion



multiple targets

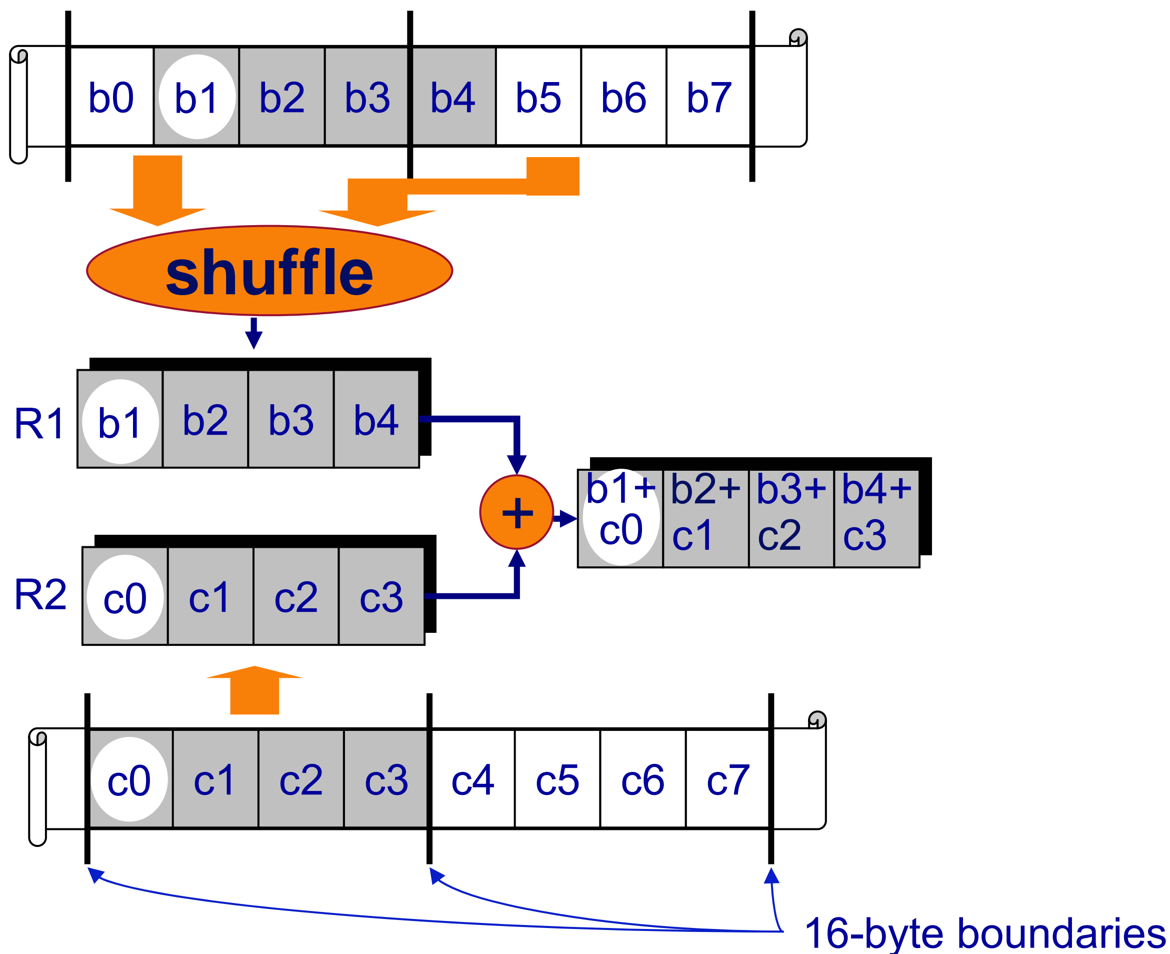


© 2006 IBM Corporation

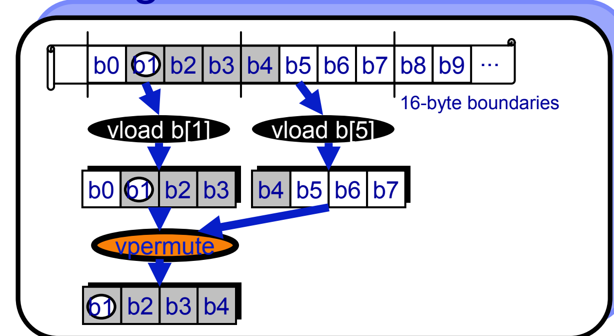
Example of SIMD Constraints (cont.)

Alignment in SIMD units matters

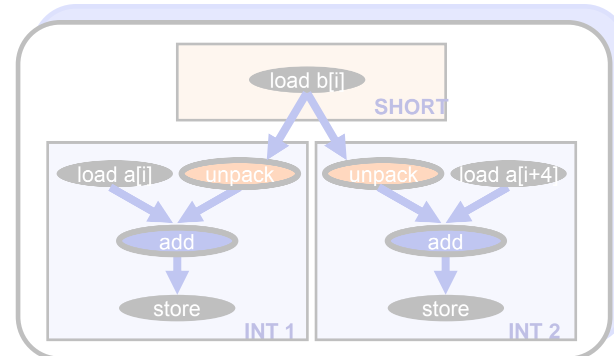
- when alignments within inputs do not match
- must realign the data



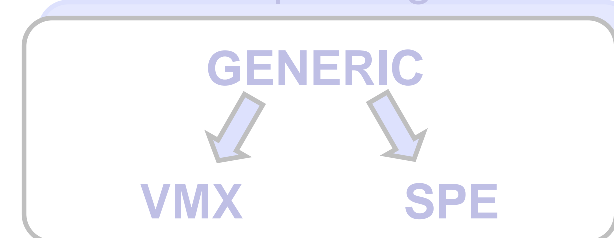
alignment constraints



data size conversion



multiple targets

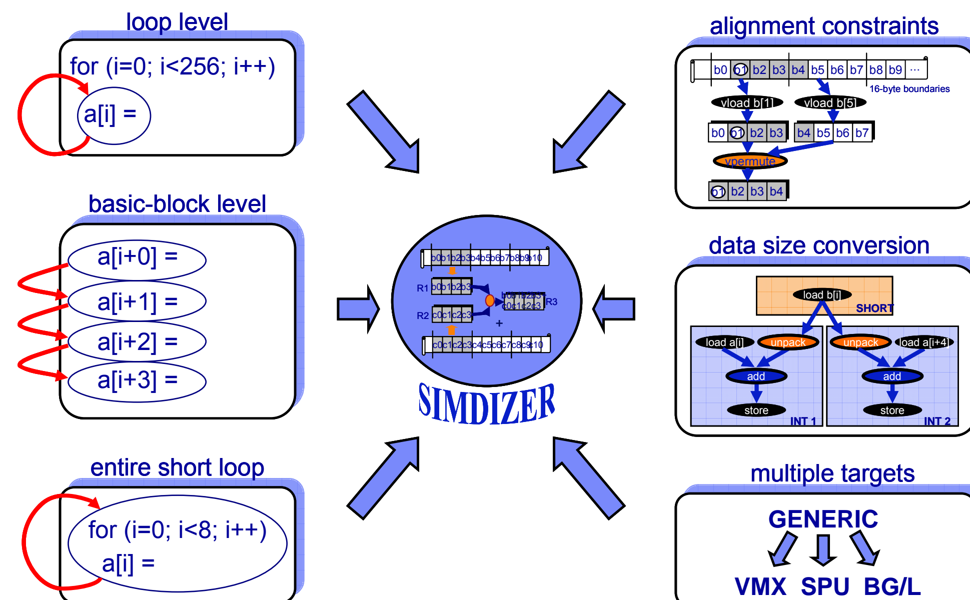


© 2006 IBM Corporation

Automatic Simdization for Cell

❑ Integrated Approach

- extract at multiple levels
- satisfy all SIMD constraints
- use “virtual SIMD vector” as glue



❑ Minimize alignment overhead

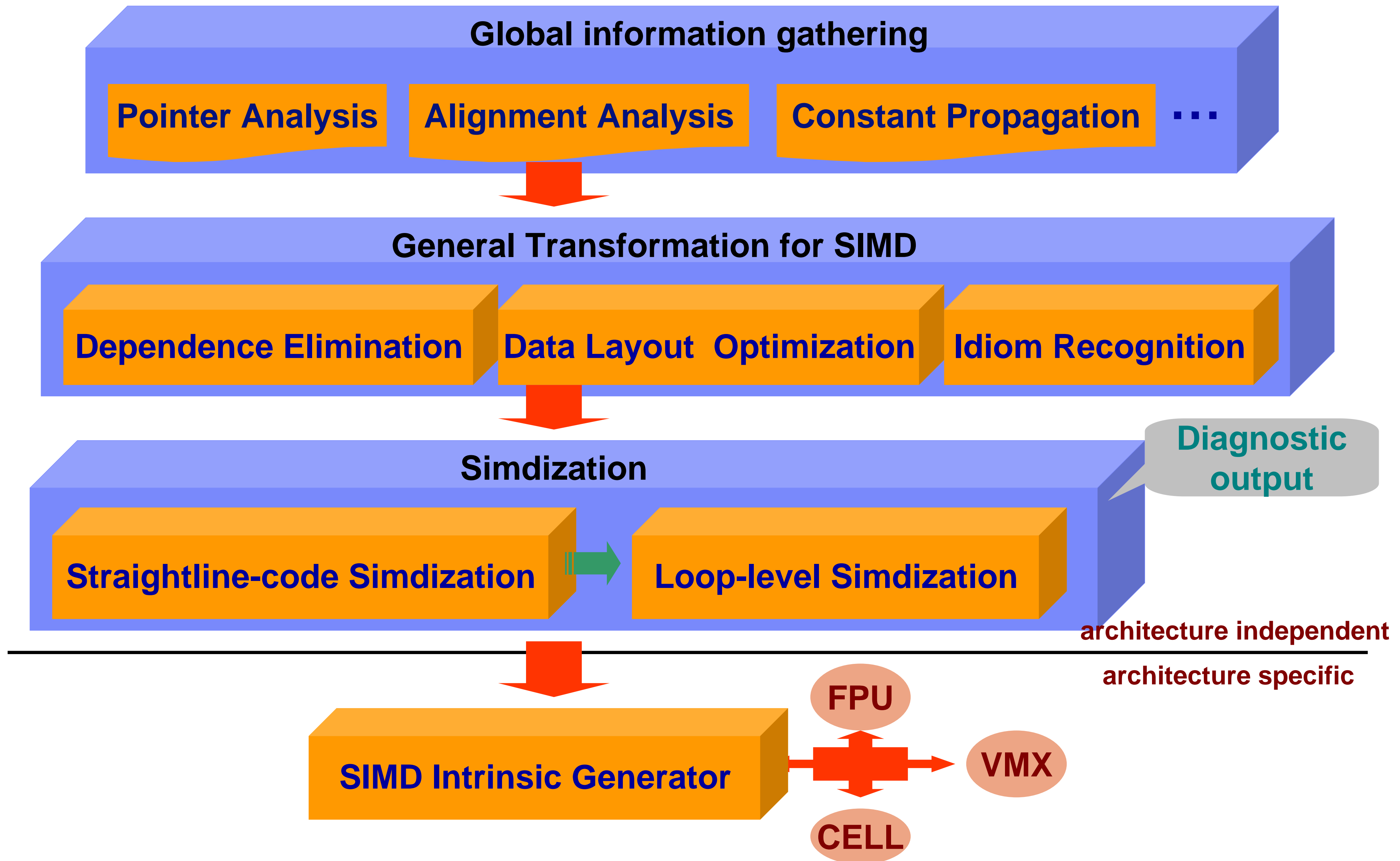
- lazily insert data reorganization
- handle compile time & runtime alignment
- simdize prologue/epilogue for SPEs
  - memory accesses are always safe on SPE

❑ Full throughput computations

- even in presence of data conversions
- manually unrolled loops...

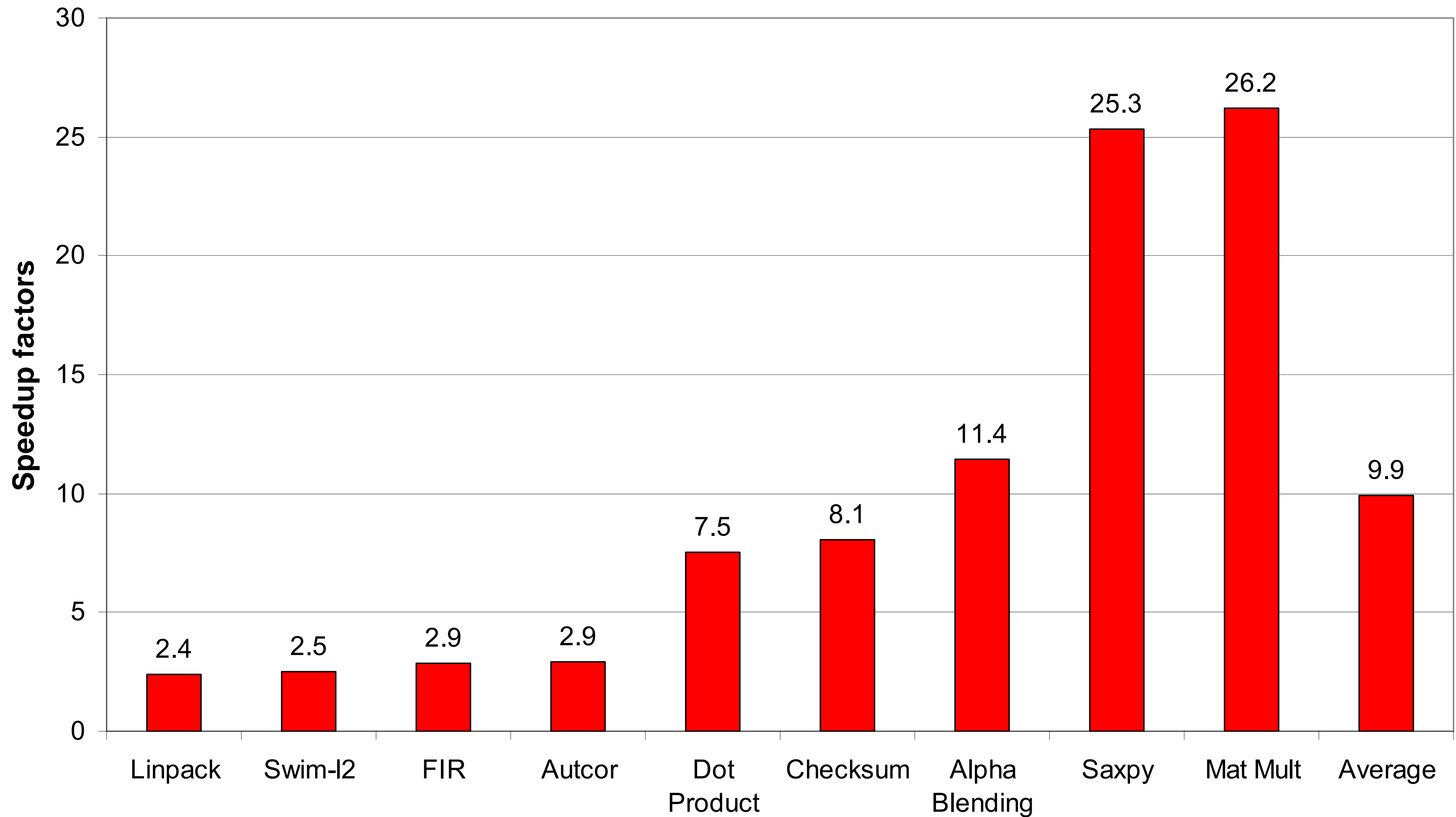
© 2006 IBM Corporation

A Unified Simdization Framework



© 2006 IBM Corporation

SPE Simdization Results



single SPE, optimized, automatic simdization vs. scalar code

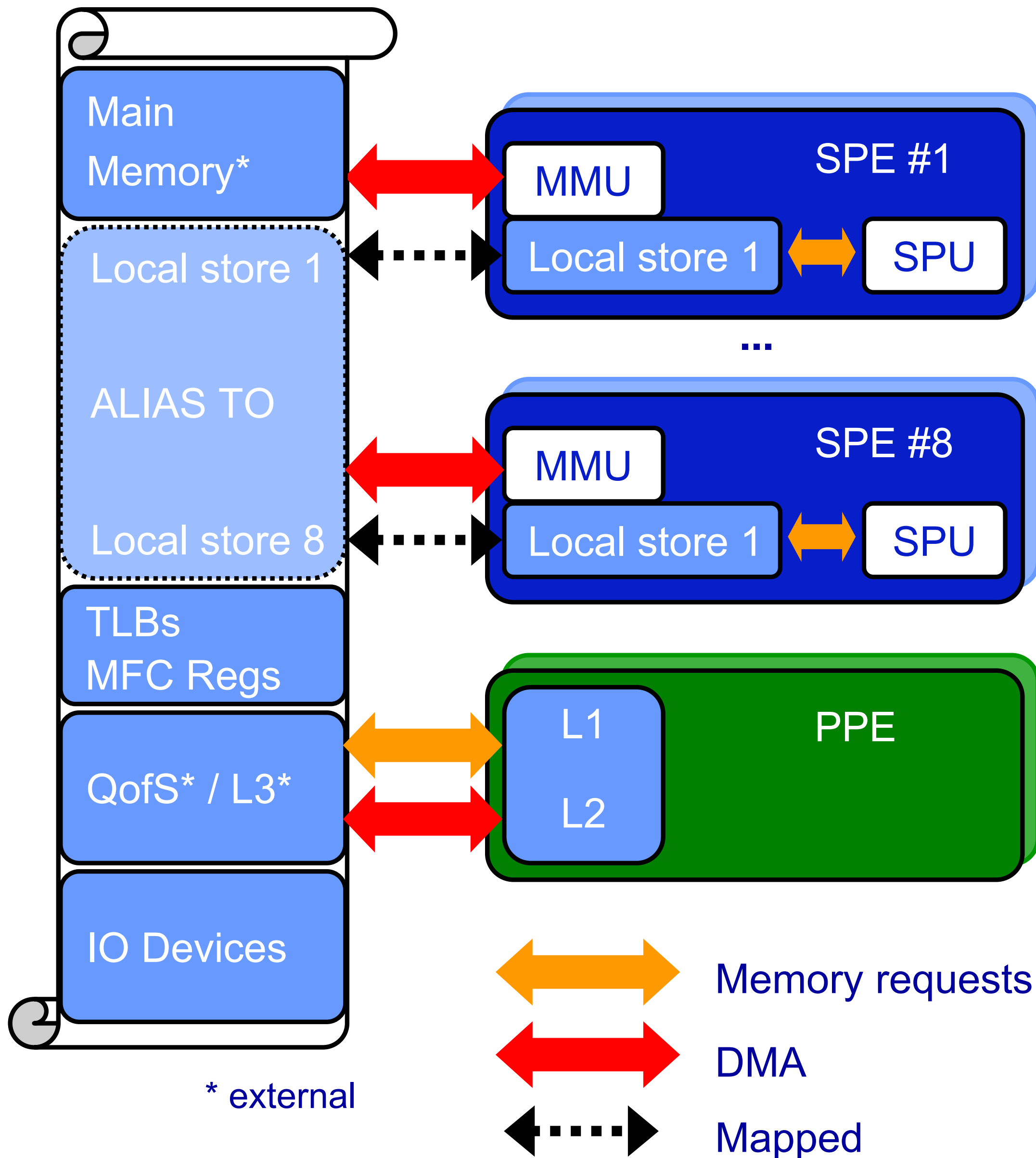
© 2006 IBM Corporation

Outline

Part 1: Automatic SPE tuning	Part 2: Automatic simdization	Part 3: Shared memory & Single program abstr.
Multiple-ISA hand-tuned programs	Explicit SIMD coding	Explicit parallelization with local memories
PROGRAMS	SIMD	PARALLELIZATION
	SIMD/alignment directives	Shared memory, single program abstraction
Automatic tuning for each ISA	Automatic simdization	Automatic parallelization

© 2006 IBM Corporation

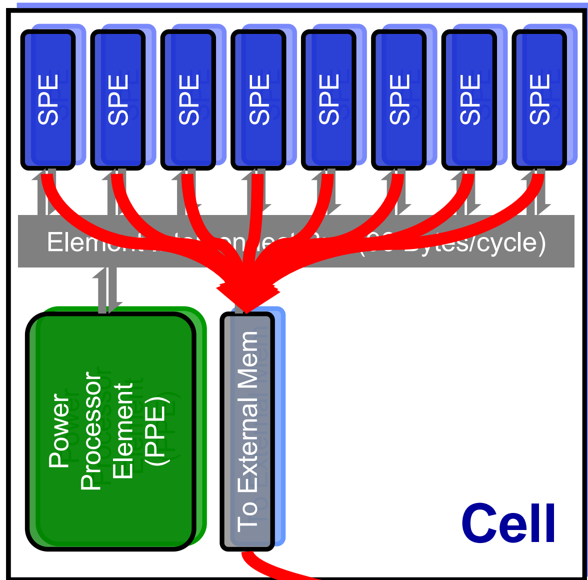
Cell Memory & DMA Architecture



- **Local stores are mapped in global address space**
- **PPE**
  - can access/DMA memory
  - set access rights
- **SPE can initiate DMAs**
  - to any global addresses,
  - including local stores of others.
  - translation done by MMU
- **Note**
  - all elements may be masters, there are no designated slaves

© 2006 IBM Corporation

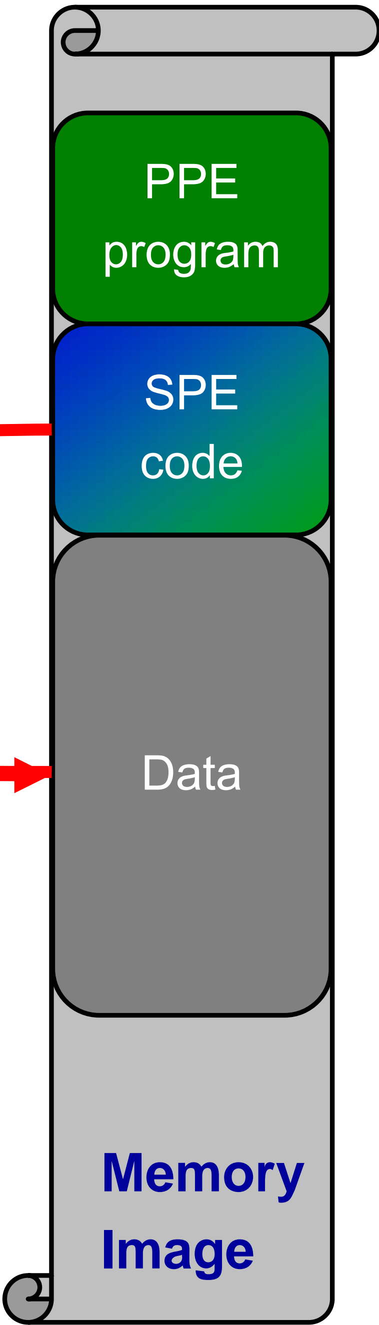
Anatomy of a Cell Program



for (i=0; i<10K; i++)

A[i] = B[i] + C[i]

Program



A. Invoke PPE program

A1. Invoke thread lib to start threads

A2. Load SPE code “loop1” and initiate

A3. Wait for SPE to finish

B. SPE code “loop(lb, ub)

B1. dma_get B,C[lb : ub];

B2. for (i=lb; i<ub; i++)

A[i] = B[i] + C[i];

B3. dma_put A[lb : ub];

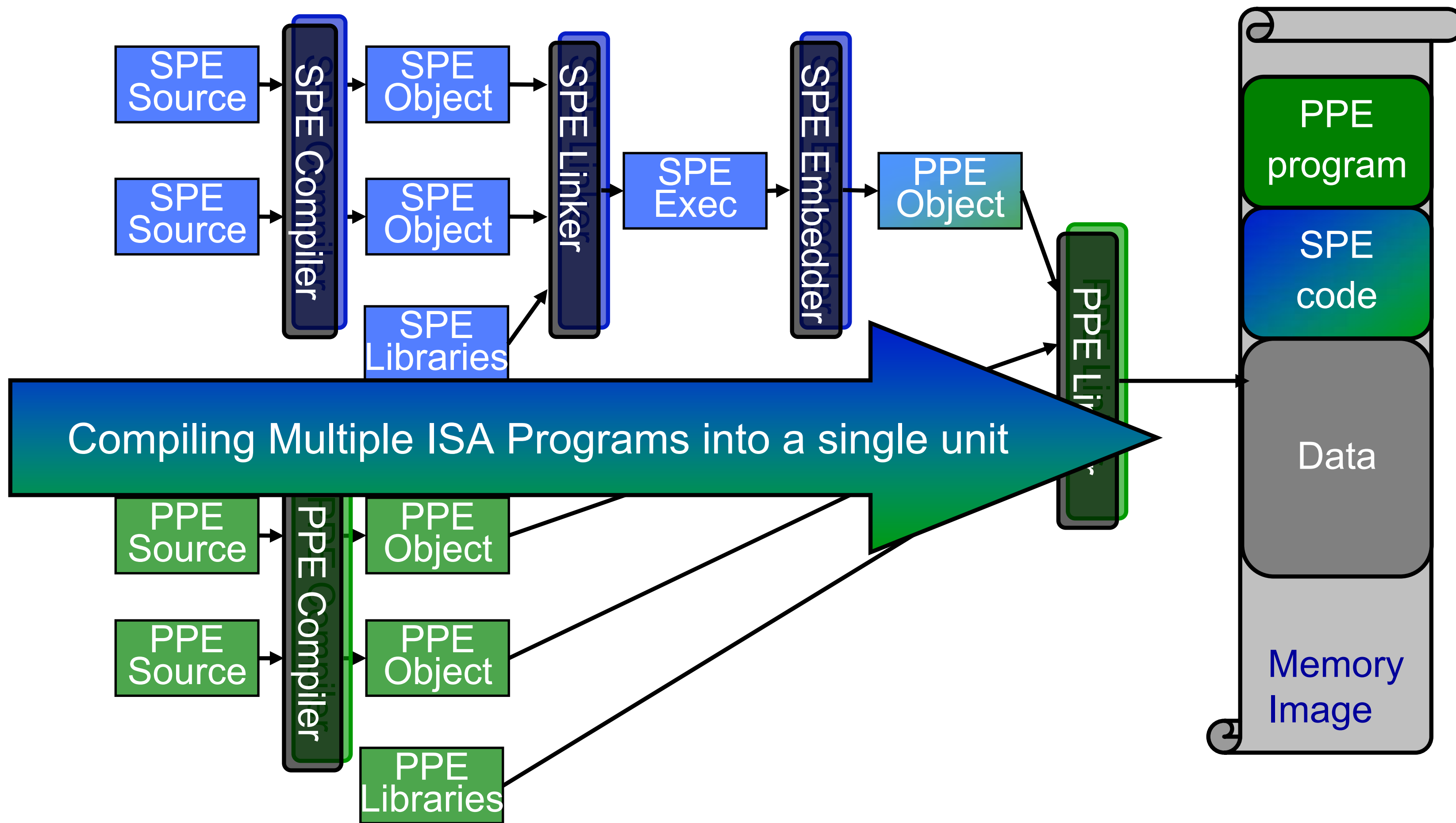
© 2006 IBM Corporation

Manual Compilation of a Cell Program

PPE/SPE Progs. **Manual Compiling & Binding** **Executable**

B1. dma_get B,C[lb : ub];
B2. for (i=lb; i<ub; i++)
A[i] = B[i] + C[i];
B3. dma_put A[lb : ub];

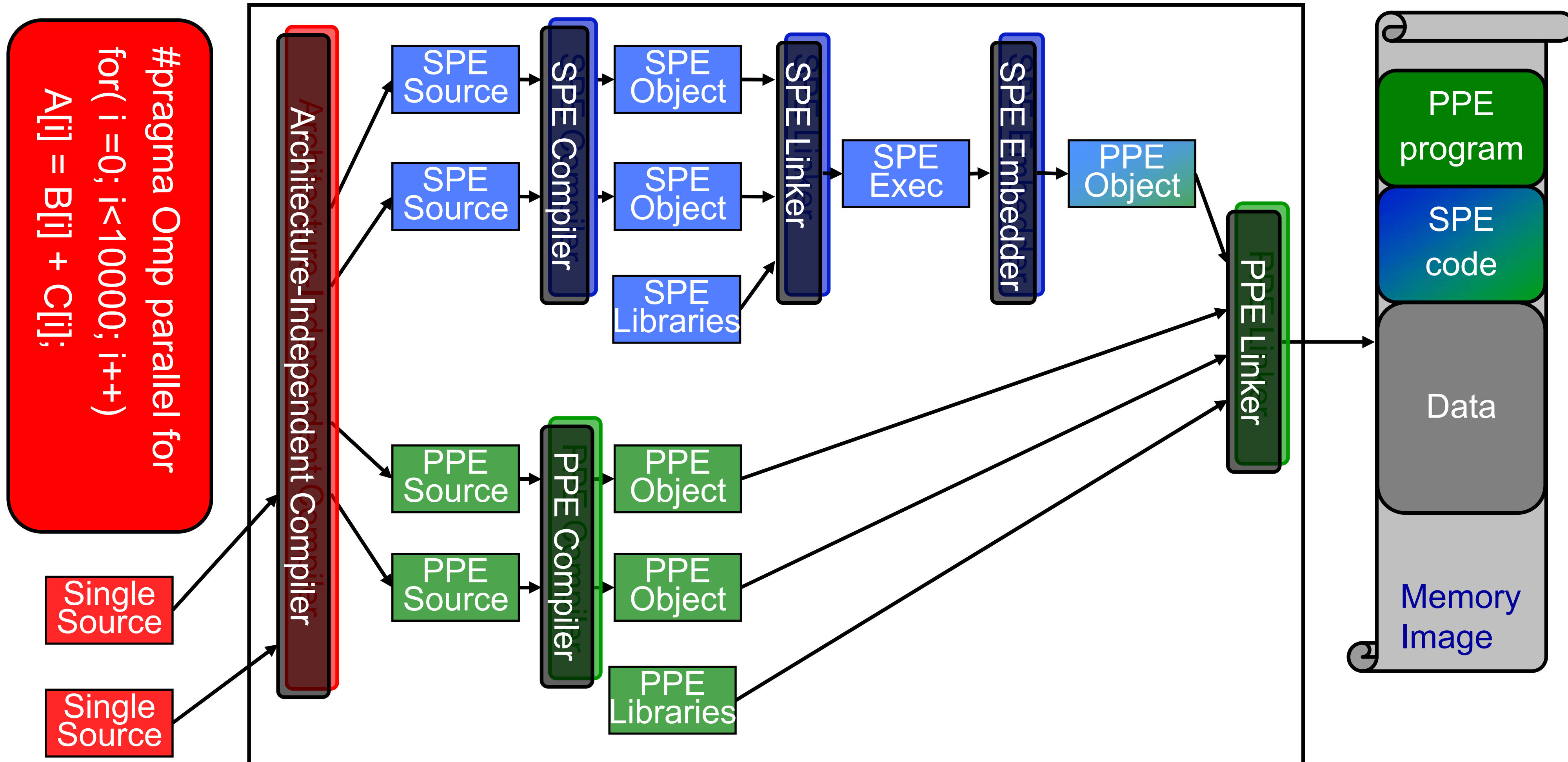
A1. Invoke thread lib to start threads
A2. Load SPE code "loop1" and initiate
A3. Wait for SPE to finish



Highest performance: use platform specific instrinsics
Higher productivity using automatic simdization compiler

© 2006 IBM Corporation

"Single-Source" Compilation of a Cell Program



Highest productivity using single-source compiler

© 2006 IBM Corporation

Parallelization and Partitioning Approach

- **Parallel systems present a complex programming task**

- **Heterogeneous parallel systems increase the complexity because …**

- **Must consider processor characteristics in addition to parallelism**
  - different processor performance
  - different system services (no O/S on SPEs)
  - different memory latencies
  - small local memories
  - to what extent is the PPE involved?
  - concept that PPE is only a service provider, most computation takes place on SPE

© 2006 IBM Corporation

Our Parallel Implementation

- **Avoid pitfalls of fully automatic approach by ...**
- **Allowing user to guide partitioning with directives, initially OpenMP**
  - reasonable acceptance of OpenMP in the parallel programming community
  - directives and clauses easily mapped to SPE parallel execution
  - parallel_for, parallel_sections (possible pipelining functionality), parallel regions etc
  - relaxed-consistency memory-model incorporated into software cache approach
- **Provide solution for C and Fortran**
- **Sidesteps need to partition for code size, at least initially**
  - since most loop bodies typically fit within SPE storage
- **Roadmap to fully automatic approaches**
  - can piggyback on compilers that automatically generate OpenMP directives
  - XL compiler has an initial version of automatic OpenMP directive generation

© 2006 IBM Corporation

"Single-Source" Compiler

- **User prepares an application as a collection of one or more source files containing user directives/pragmas**

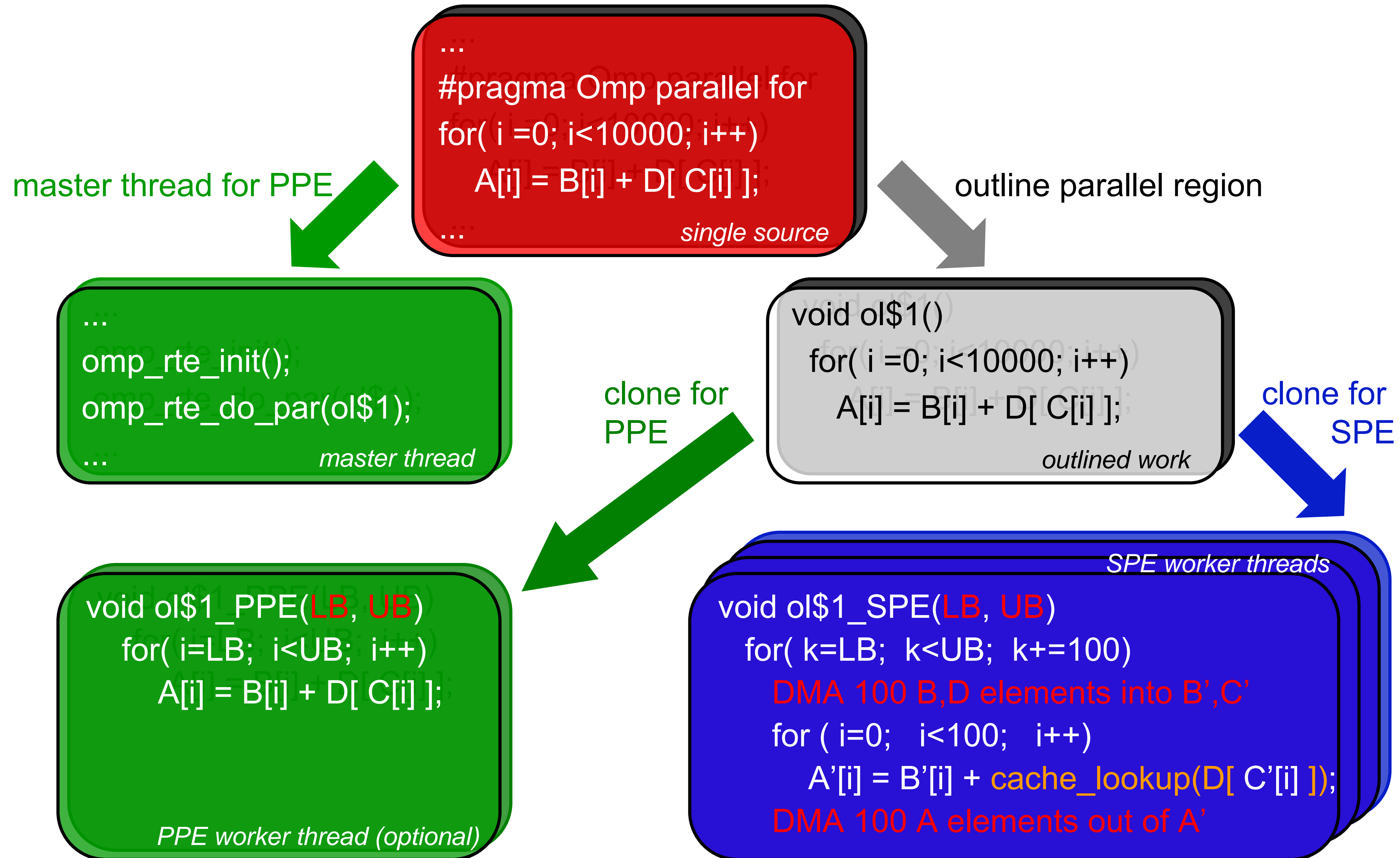
- **Compiler uses directives to partition code between PPE and SPE**

- **Compiler handles data transfers.**
  - identify accesses in SPE functions that refer to data in system memory locations
  - use static buffers or software cache to transfer his data to/from SPE local stores

- **Compiler handles code size**
  - explore extending Code partitioning to Single Source, i.e. automatic partitioning based on functionality rather than size

© 2006 IBM Corporation

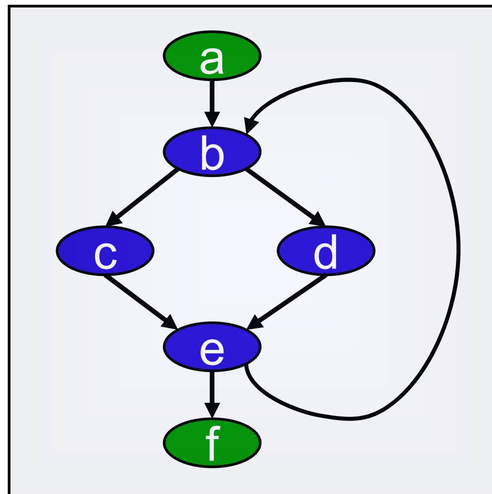
Single-Source Compiler using OpenMP pragma



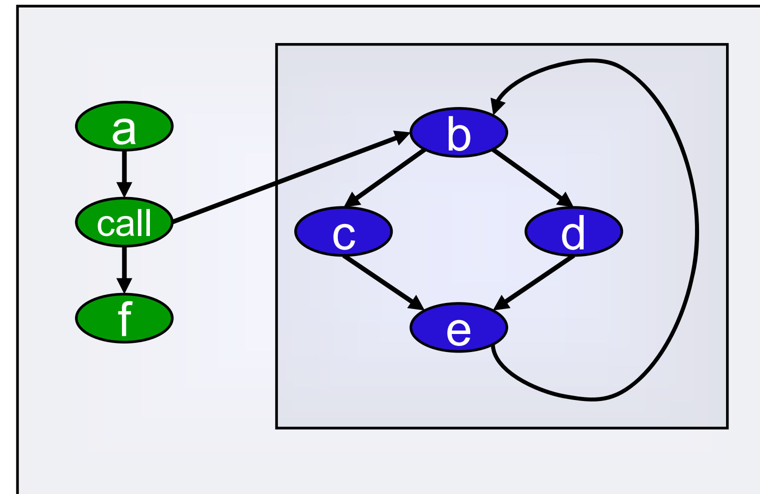
© 2006 IBM Corporation

Outlining a Parallel Region

- **Creates a new architecture-independent function**
  - step proceeds in Pass 1 of TPO
  - add an extra node in the call graph
  - cannot have jumps outside region



outline



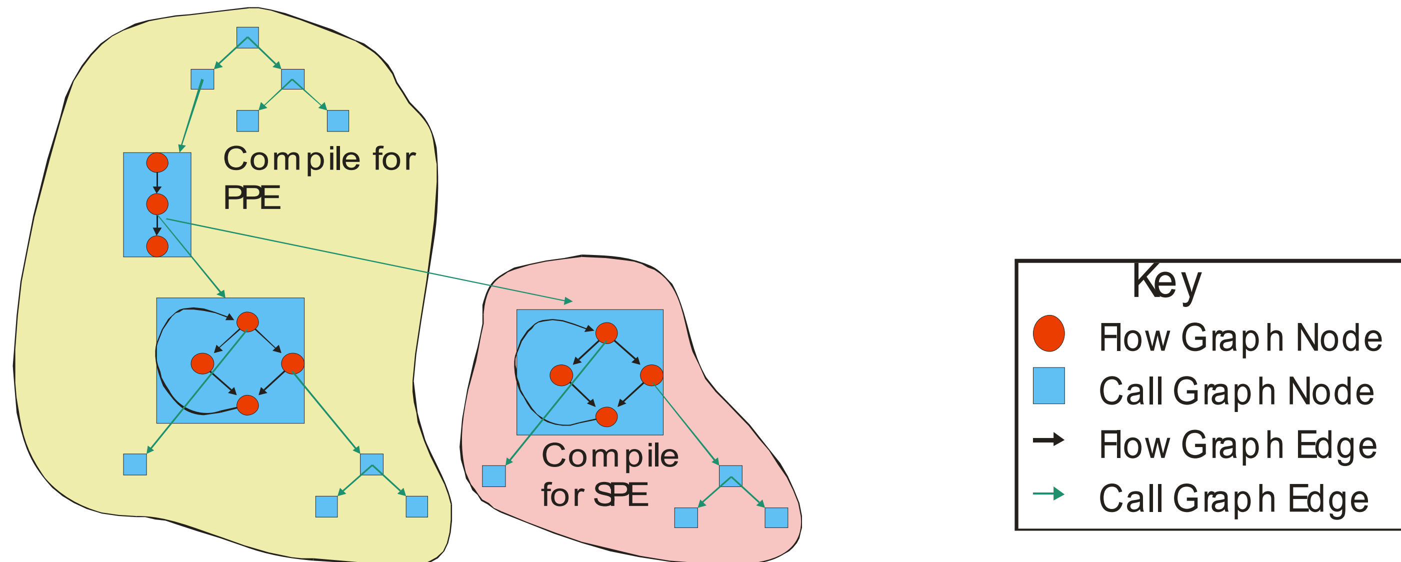
control flow graphs & function scope

  - procedures are nested to access shared variables
  - memory alias sets must accurately represent use/defs
  - private variables are transformed to automatic in the new procedure; must transfer alias relationships

© 2006 IBM Corporation

Cloning and Specializing a Parallel Region

- **Create new architecture-dependent functions:**
  - step proceeds in Pass 2 of TPO (uses whole program call graph)
  - outlined function is cloned and then specialized to create a PPE and an SPE version



- **All called functions must also be cloned**
  - SPE call sites modified to call SPE versions of cloned subroutines

- **Compiler creates SPE and PPE partitions**
  - invokes lower-level optimizer for each partition to perform machine specific optimization

© 2006 IBM Corporation

PPE Runtime

- **First OMP construct initializes the runtime system**
  - create SPE threads and loads the SPE runtime
  - create work queue and get DMA queue addresses
  - send address of work queue to each SPE
  - set global options
- **Sends a “setup_done” to SPEs after partitioning/scheduling the work items**
- **Calls PPE outlined procedures for its own work share.**

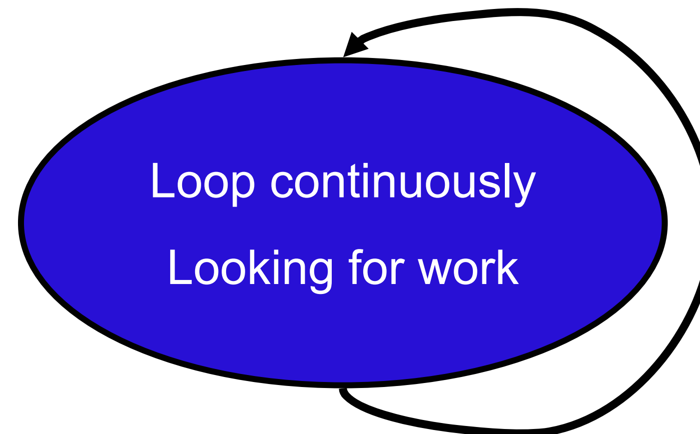
```
...
omp_rte_init();
omp_rte_do_par(ol$1);
...
```
master thread

```
void ol$1_PPE(LB, UB)
    for( i=LB;  i<UB;  i++)
        A[i] = B[i] + D[ C[i] ];
```
PPE worker thread

© 2006 IBM Corporation

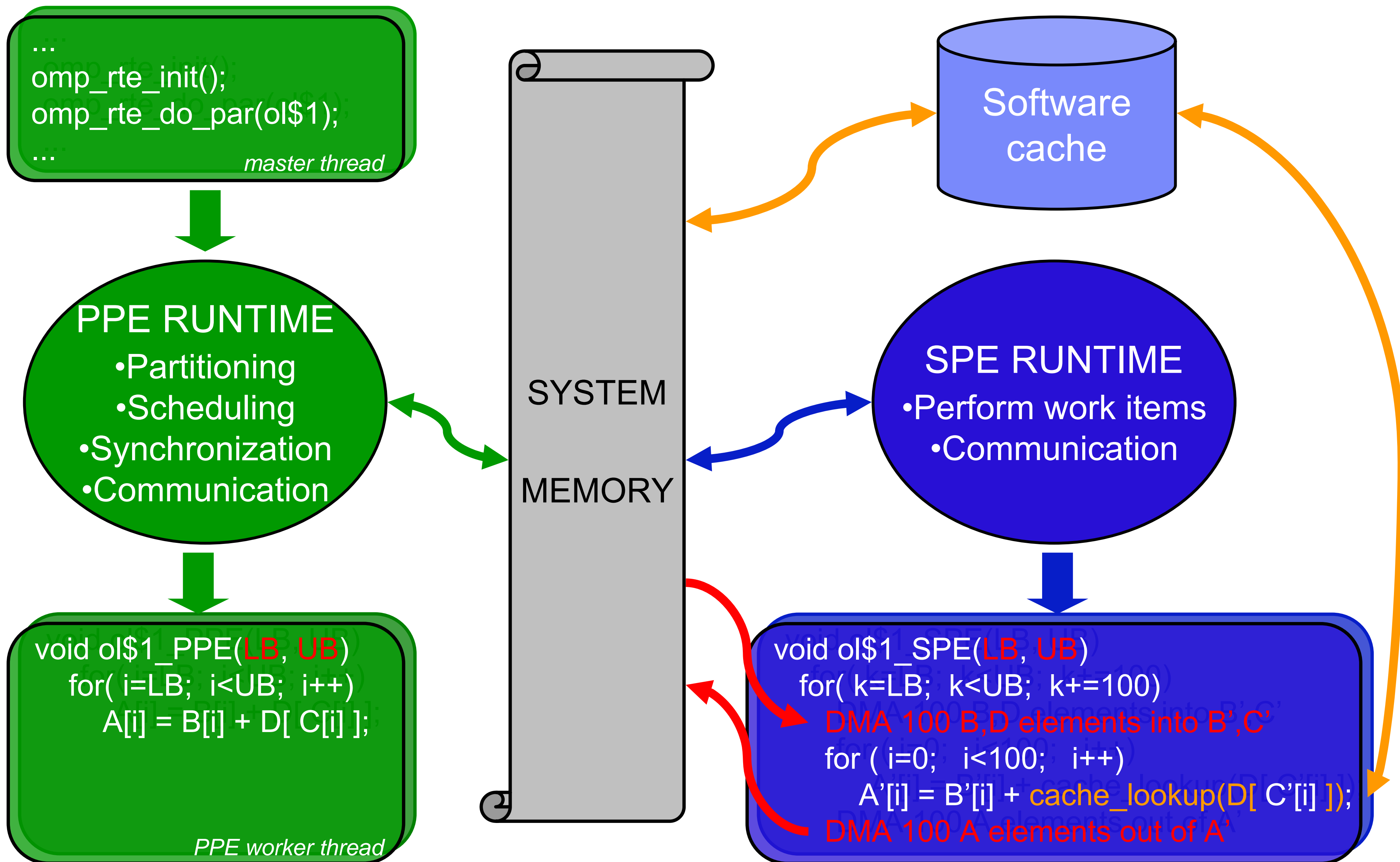
SPE Runtime

- **Infinite loop while waiting for signals from PPE runtime**
- **DMA-fetches work items from work queue in system memory**
- **Depending on the work type:**
  - translates the address of SPE outlined procedure from PPE outlined procedure
  - invokes SPE outlined procedures.



© 2006 IBM Corporation

Runtime Interaction



© 2006 IBM Corporation

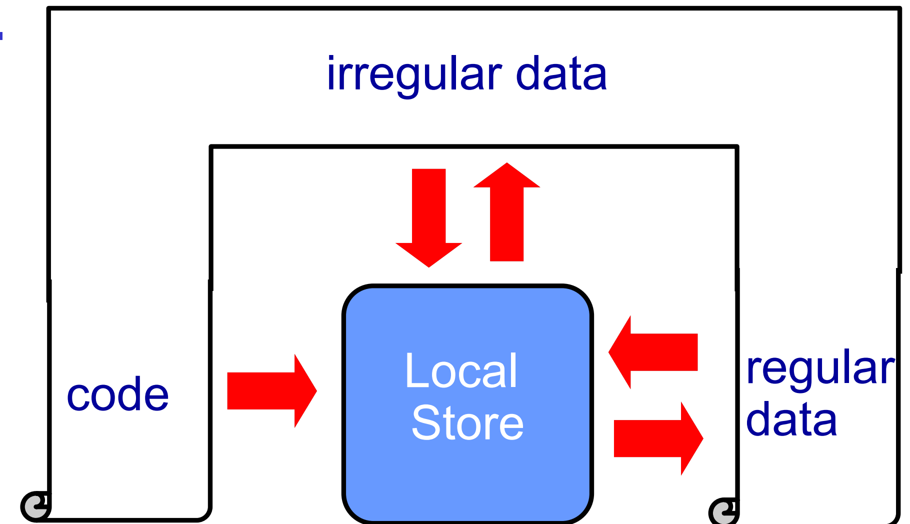
Limitations

- **No nested parallelism**
- **Parallel constructs with system calls serialized (execute on PPE)**
- **Aggregated SPE binary not partitioned yet**
- **Some less-frequently used features under development**

© 2006 IBM Corporation

Competing for the SPE Local-Store

Local store is fast, need support when full.

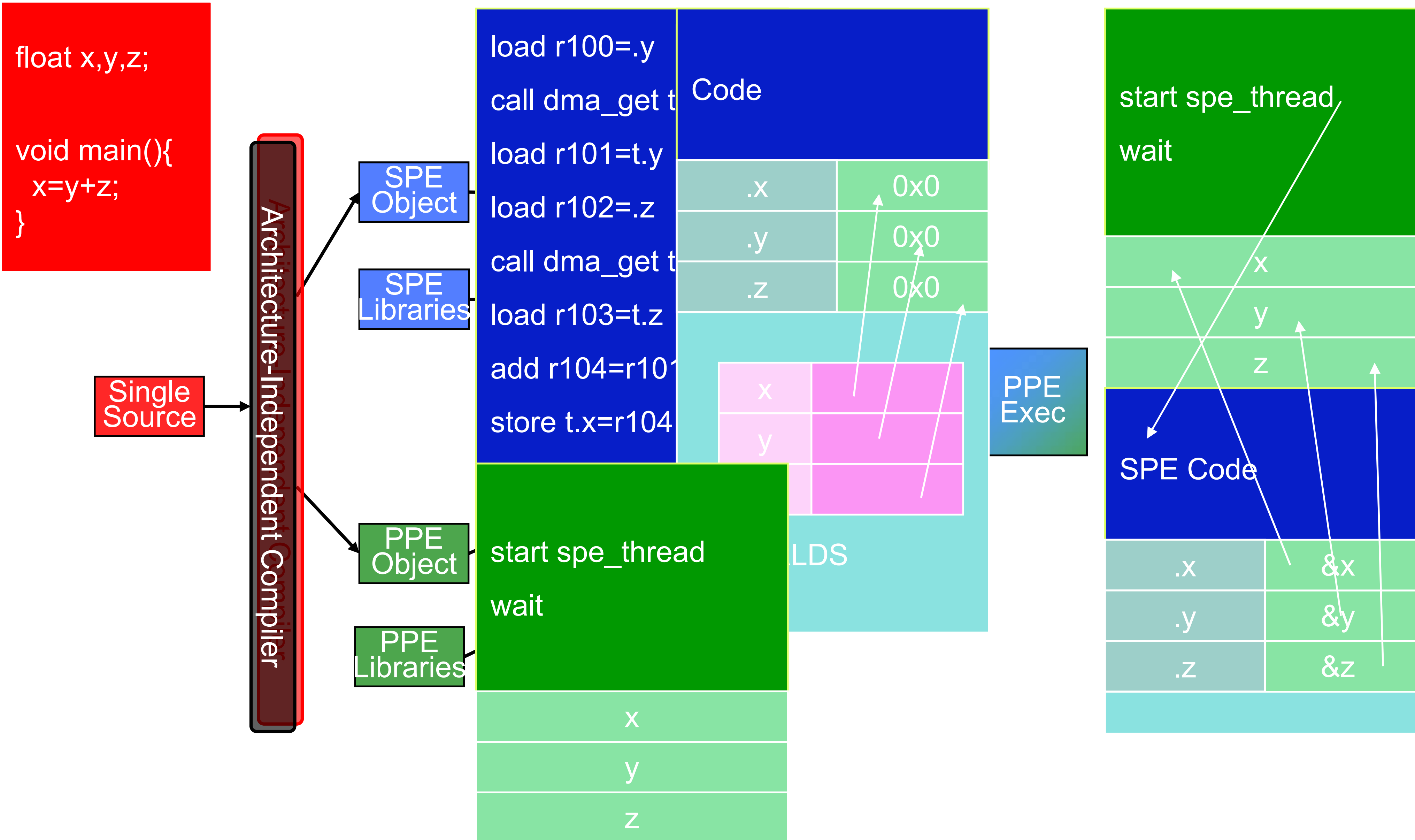


Provided compiler support:

- **SPE code too large**
  - compiler partitions code
  - partition manager pulls in code as needed
- **Data with regular accesses is too large**
  - compiler stages data in & out
  - using static buffering
  - can hide latencies by using double buffering
- **Data with irregular accesses is present**
  - e.g. indirection, runtime pointers...
  - use a software cache approach to pull the data in & out (last resort solution)

© 2006 IBM Corporation

Accessing Global Variables in System Memory



© 2006 IBM Corporation

Hiding Communication using Double Buffering

Original Code

```
for( i=0; i<100000; i++)
    A[i] = B[i] + C[i];
```

Single Buffering

```
for( i=0; i<100000; i+=100)
    dma_get(B', B[i], 400);
    dma_get(C', C[i], 400);
    for( ii=0; ii<100; ii++)
        A'[ii] = B'[ii] + C'[ii];
    dma_put(A[i], A', 400);
```

Double Buffering

```
dma_get(B', B[0], 400);
dma_get(C', C[0], 400);
for(i=0;i<99800;i+=200)
    dma_get(B", B[i+100], 400);
    dma_get(C", C[i+100], 400);
    for( ii=0; ii<100; ii++)
        A'[ii] = B'[ii] + C'[ii];
    dma_put(A[i],A',400);
    dma_get(B', B[i+200], 400);
    dma_get(C', C[i+200], 400);
    for( ii=100; ii<200; ii++)
        A"[ii] = B"[ii] + C"[ii];
     dma_put(A[i+100], A",400);
for(ii=0;ii<100;ii++)
    A'[ii] = B'[ii] + C'[ii];
dma_put(A[i+99900], A', 400);
```

communication is blocked (100 elements at a time)

computation and communication overlap as their phases are software pipelined

© 2006 IBM Corporation

Handling Irregular Accesses using Software Cache

Original Code

```
for(i=0;i<100000;i++)
    = … D[ C[i] ];
```

Code with explicit Cache Lookup

```
for(i=0;i<100000;i++)
    t=cache_lookup( D[ C[i] ]);
    = … t;
```

Code Lookup Sequence

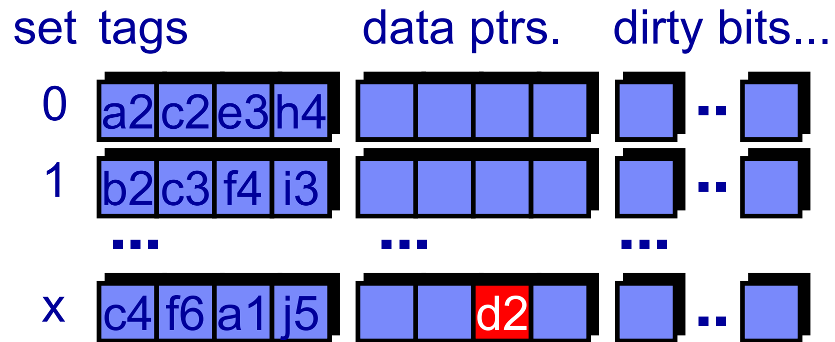
```
inline vector cache_lookup (addr)
    if (cache_directory[addr&key_mask] != (addr&tag_mask))
        miss_handler(addr);
    return cache_data[addr&key_mask][addr&offset_mask];
```

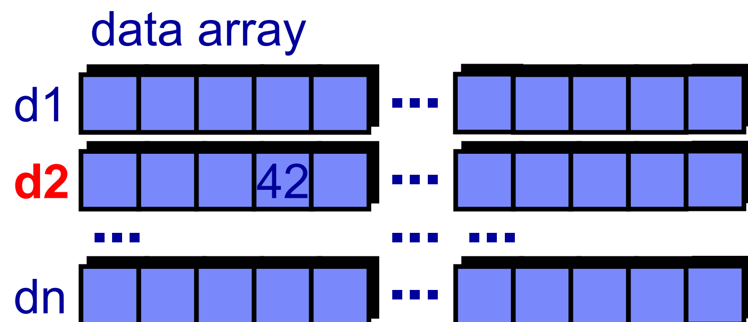
miss handler DMA the required data, and some suitable quantity of surrounding data

higher degrees of associativity can be supported, for little extra cost on a SIMD processor

© 2006 IBM Corporation

Software Cache Architecture

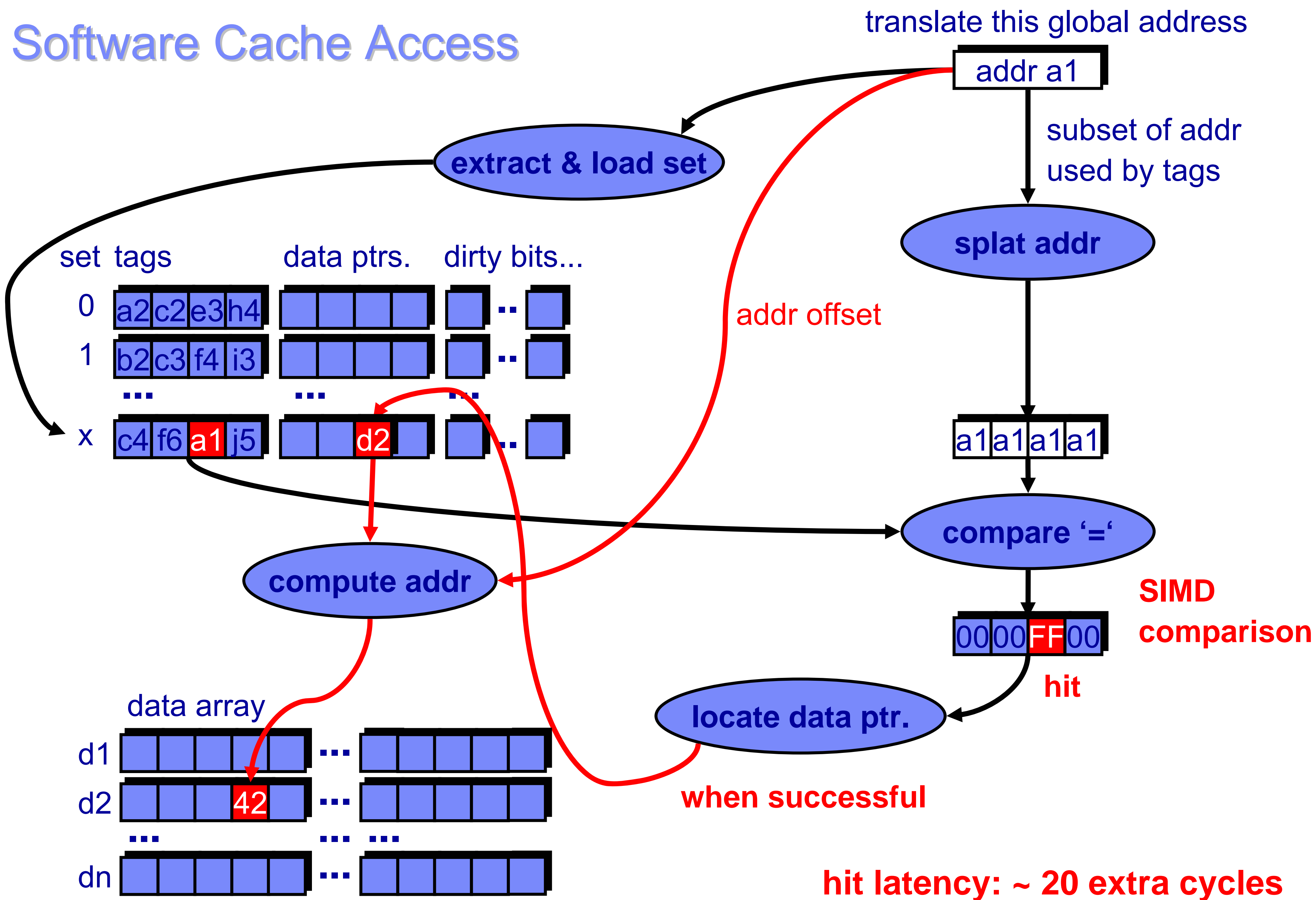




- **Cache directory**
  - 128-set, 4-way set associative
  - pointers to data lines
  - use 16KByte of data
- **Data in a separate structure**
  - 512 x 128B lines
  - use 64KByte of data

© 2006 IBM Corporation

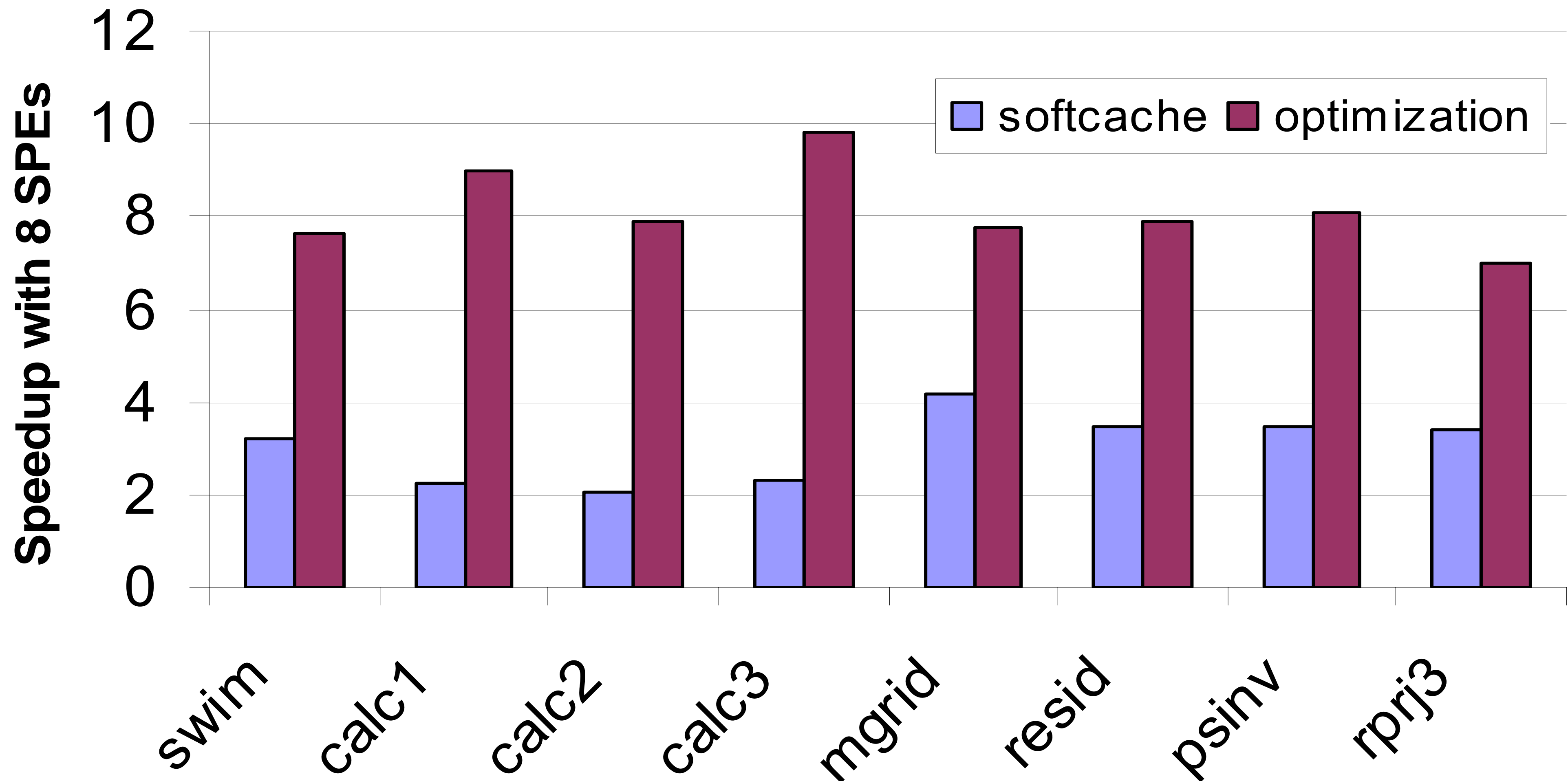
Software Cache Access



© 2006 IBM Corporation

Single Source Compiler Results

- Results for whole Swim, Mgrid, and some of their kernels



baseline: execution on one single PPE

© 2006 IBM Corporation

Conclusions

- **Cell Broadband Engine architecture**
  - heterogeneous parallelism
  - dense compute architecture
- **Present the application writer with a wide range of tool**
  - from support to extract maximum performance
  - to support to achieve maximum productivity with automatic tools
- **Shown respectable speedups**
  - using automatic tuning, simdization, and support for shared-memory abstraction

© 2006 IBM Corporation

Questions

- http://www.alphaworks.ibm.com/tech/cellcompiler

For additional info:

www.research.ibm.com/cellcompiler

© 2006 IBM Corporation