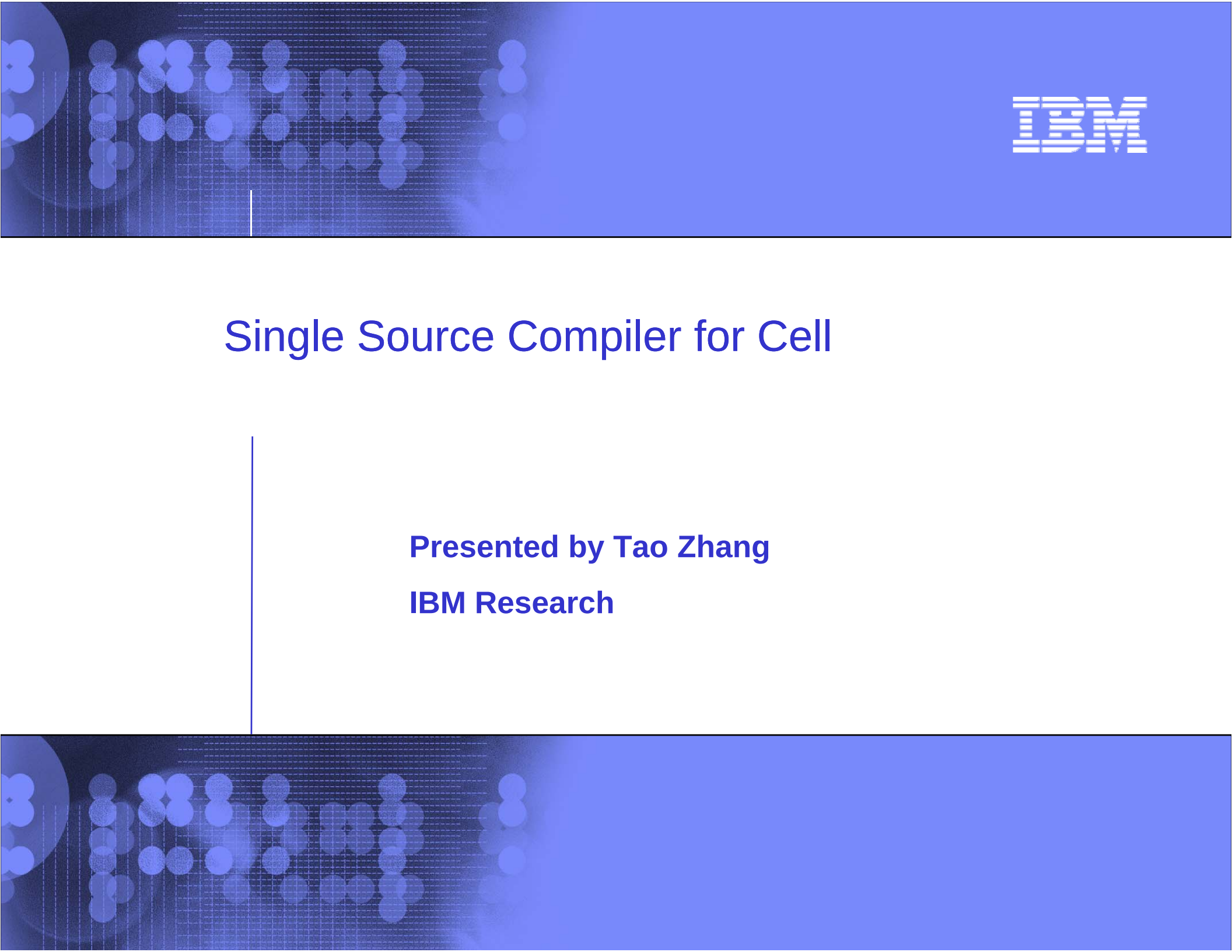


Single Source Compiler for Cell

Presented by Tao Zhang

IBM Research

Outline

- **Compiler Overview**
- **Compiler in detail**
  - Code Generation
  - Thread and Synchronization
  - Data Management
- **Conclusion**

Tasks to Exploit Heterogeneous Multi-Cores in Cell

- **Partition application into PPE and SPE portions**
  - SPE has very limited local memory
  - Both code size problem and data size problem
- **Code and compile PPE and SPE portions separately**
  - Coding and compiling for two distinct ISAs
- **Parallelize across multiple SPEs/PPE**
- **Simdization inside a single SPE**
- **Orchestration of DMA data transfers**
- **Synchronization between SPEs/PPE**
- **Quite complicated!**

Single Source Compiler for Cell

- **Single set of Source files**
  - OpenMP directives natural choice – wide acceptance, simplicity
- **Compiler, guided by OpenMP pragmas, generate optimized code for Cell**
  - code partitioning between PPE and SPEs
  - Parallelization between SPEs and PPE
  - Auto-simdization
  - Data transfers
  - Synchronization
  - Code size
- **User interactions supported**
  - Performance tuning compiler options
  - Hand-optimized functions provided by user

Example

1. Code

```
#include <stdio.h>
#define N 512
int a[N];
int b[N];

int blether() {
    int i, j;
    #pragma omp parallel for
    for (i=0; i<N; i++) {
       b[i] = a[i] + i*1000;
    }
    return 0;
}
```

```
main() {
    int i;
    int step;
    int sum = 0;
    for(i=0;i<N;i++) {
        a[i]=i;
    }

    blether();

    for(i=0;i<N;i++)
         sum += b[i];

     printf( "omptest output t1: %d\n", sum);

     if (sum == 130946816) {
          printf( "Correct!\n");
          return 0;
     } else {
          printf( "Error! Incorrect checksum\n");
          return 1;
     }

}
```

2. Compile (simple!)

cbexlc -O5 t1.c -o t1

3. Run on Cell blade

$ ./t1
omptest output t1: 130946816
Correct!

Project road map

- **Started in 2003 at Watson**
  - Watson team: Kevin O'Brien, Kathryn O'Brien, Alexandre Eichengberger, Tong Chen, Zehra Sura, Tao Zhang, Peng Wu
- **AlphaWorks release Nov. 2006**
  - With help from Toronto compiler group (Guansong Zhang is our contact)
- **DevelopWorks release Oct. 2007**
  - Technique transfer to Toronto (Guansong Zhang, Amy Wang, …) and CRL (Haibo Lin and Tao Liu)
- **GA release expected Dec. 2008**

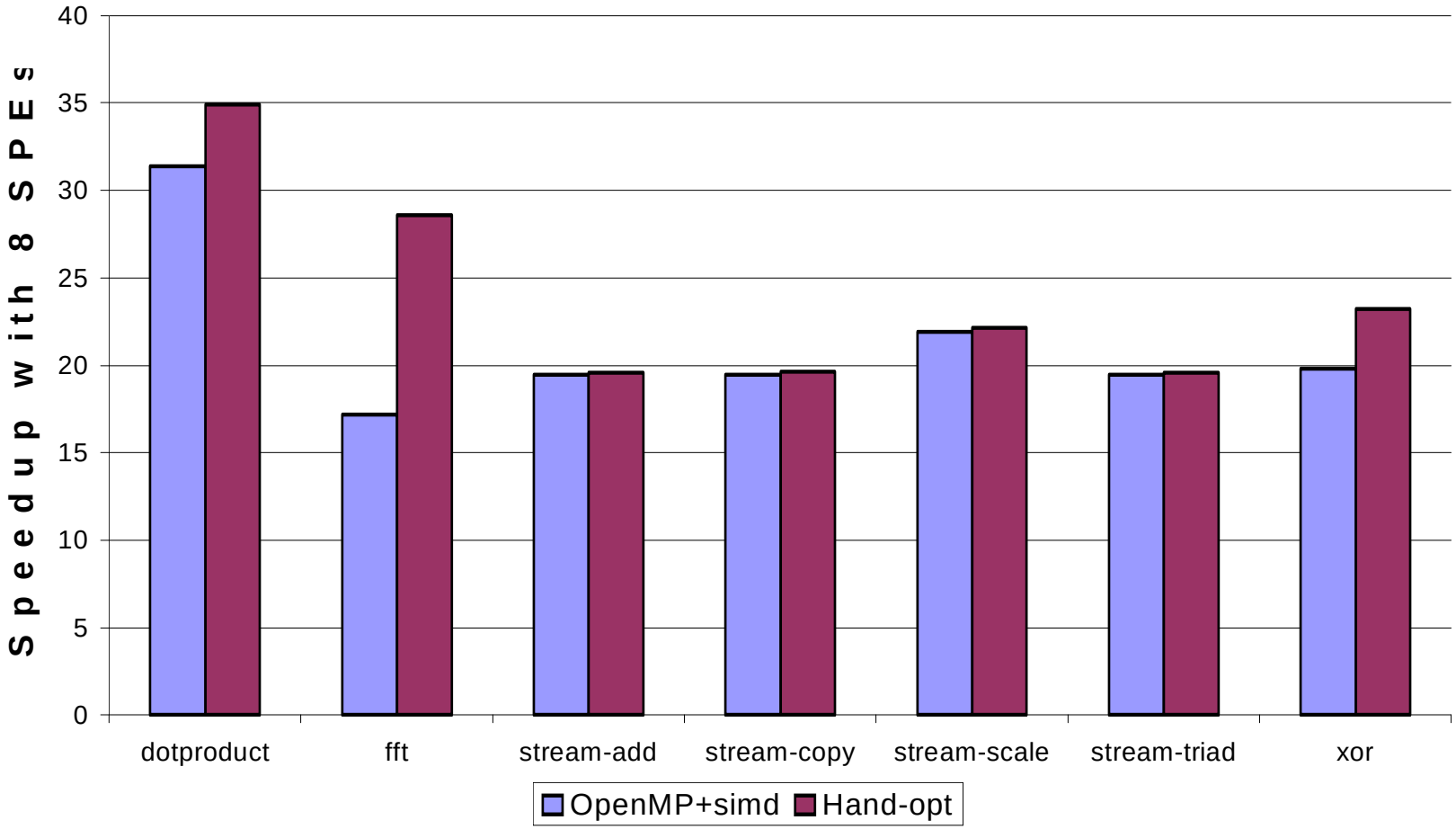
IBM XL Compiler Framework

- **IBM XL compiler already has robust infrastructure for building OpenMP compiler for Cell**
- **OpenMP and Auto parallelization currently supported in product compilers for other targets**
- **Dependence Analysis, Interprocedural Analysis, Profile Directed Feedback**
- **Aggressive Loop Optimizations and loop restructuring**
- **Function Level Partitioning to reduce compilation unit size**
- **Auto-Simdization**
- **And of course .. Supports multiple input languages and already targets a variety of different architectures**
  - PPE and SPU compiler

Components of Cell Single Source Compiler

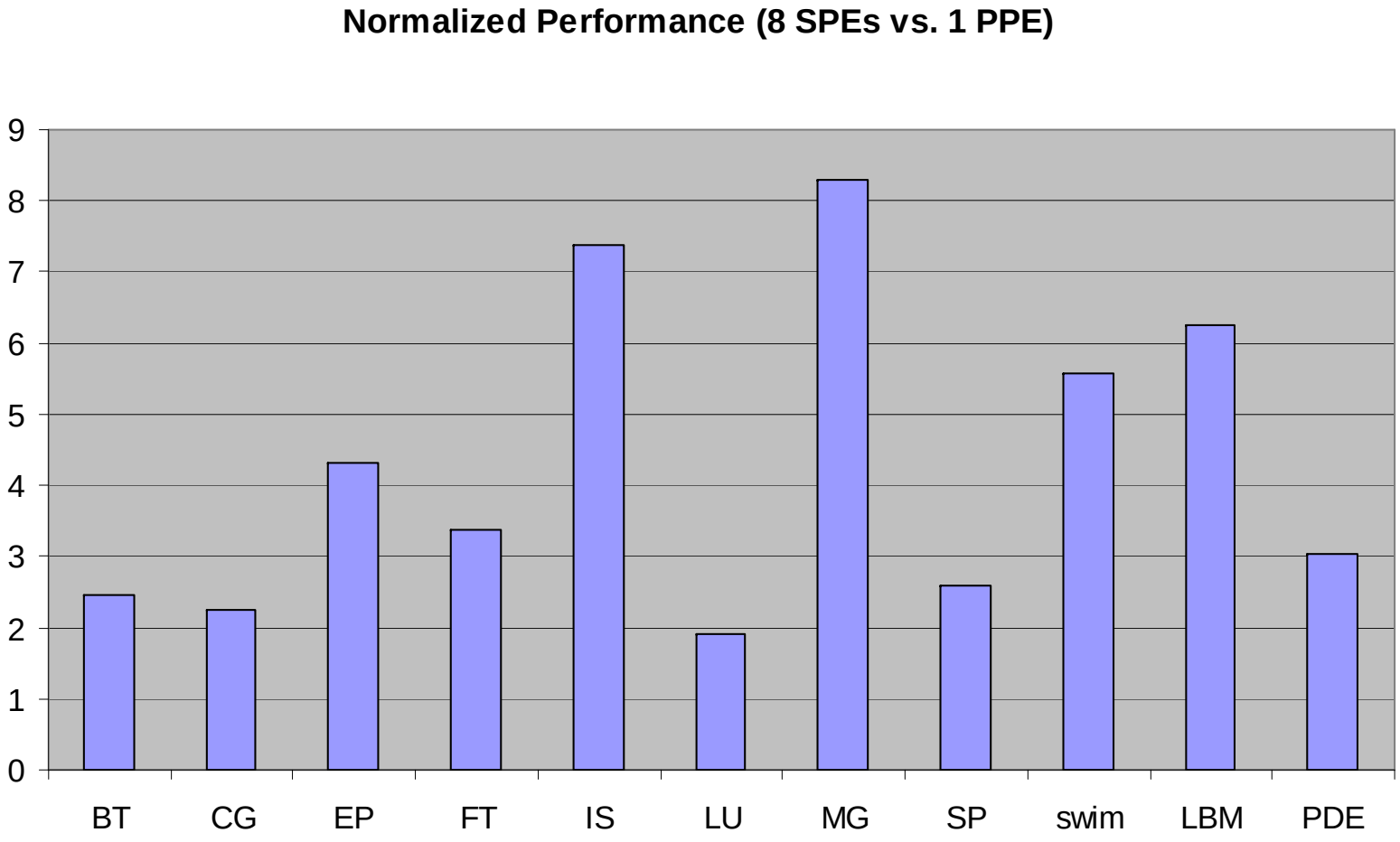
- **Compiler transformations + runtime library support**
- **Compiler**
  - Translates OpenMP pragmas in the source code
  - Implements corresponding OpenMP constructs
    - Outlines code segment enclosed in parallel constructs
    - Insert proper OpenMP runtime library calls
- **Runtime library**
  - Provides basic utilities for OpenMP on Cell
  - Thread management, work distribution, synchronization and etc

Performance of OpenMP vs. hand-optimization



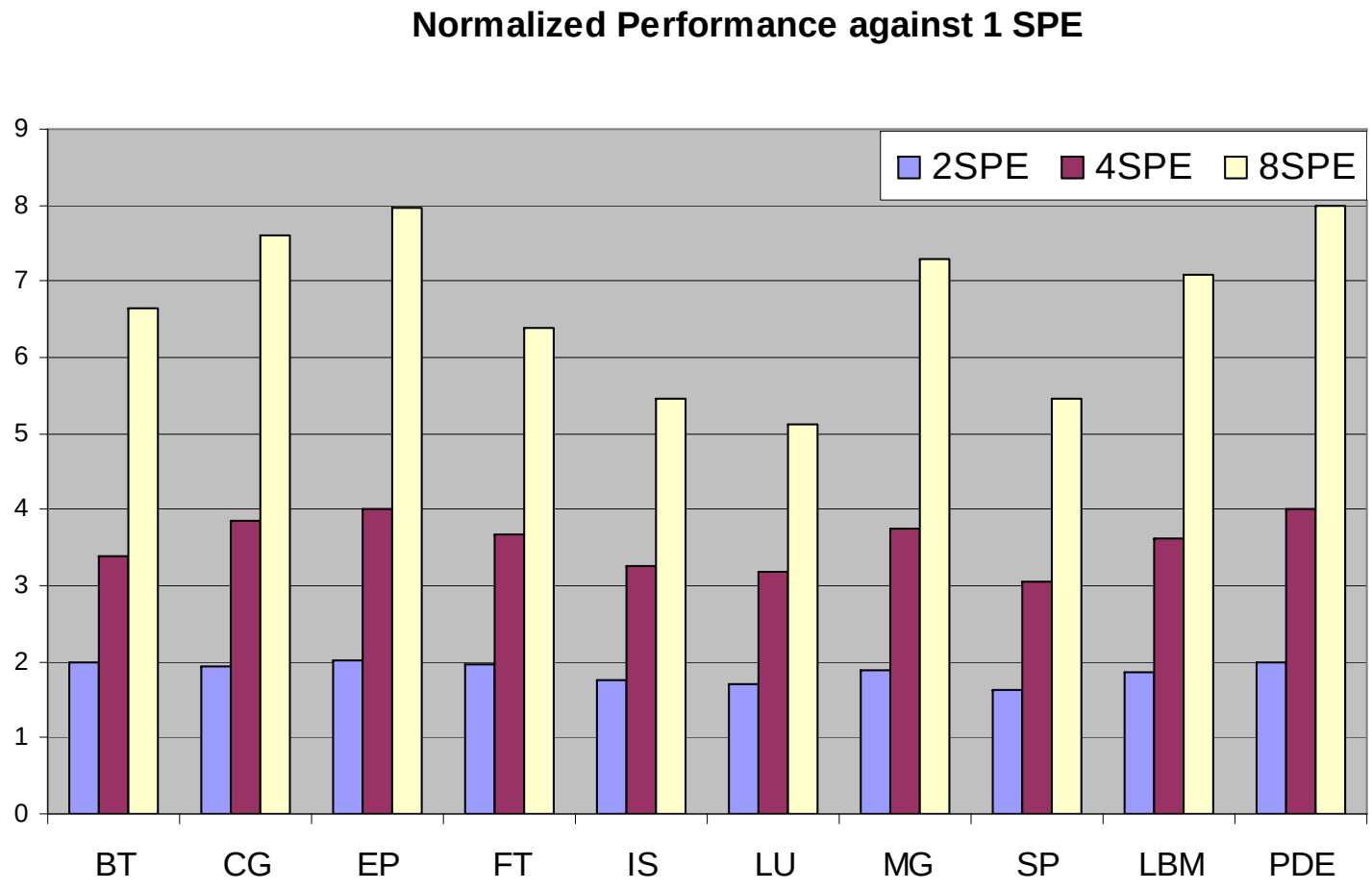
- Simple streaming benchmarks, speedups of 8 SPEs vs. 1 PPE
- Except for FFT, OpenMP compiler performs comparably against hand-optimized code

Performance on Large Benchmarks



- **QS20 machine, NAS benchmarks, one SPEC OMP2000, one SPEC2006, financial application, speedups against 1 PPE**
- **Acceptable performance on average**
- **Some results are impressive**

Scalability on Large Benchmarks



- QS20 machine, speedups against 1 SPE
- Scalability generally very good

Outline

- **Compiler Overview**
- **Compiler in detail**
  - Code Generation
  - Thread and Synchronization
  - Data Management
- **Conclusion**

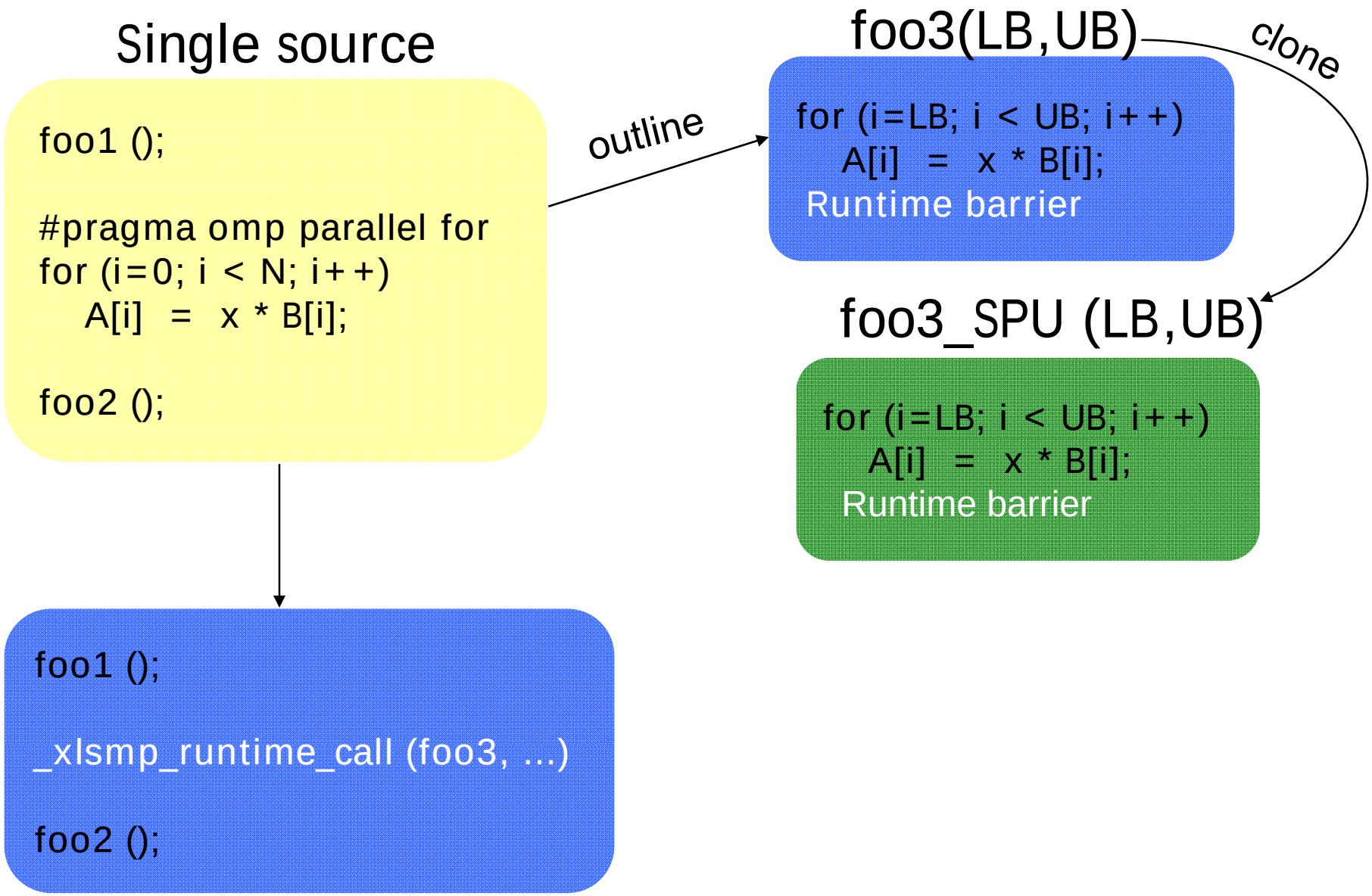
Code Generation – Outlining and Cloning

- **Outlining for parallel constructs**
- **Outlined function may be executed on both PPE and SPEs**
  - Due to the heterogeneity of Cell
- **Cloning the outlined function, one copy for PPE and one copy for SPE**
  - Enable architecture specific optimizations later
- **Functions for PPE and SPE are separated into different compilation units and different backends are invoked**
  - PPE and SPE object files generated respectively
  - After SPE binary generated, it is embedded and linked with other PPE objects into final PPE binary

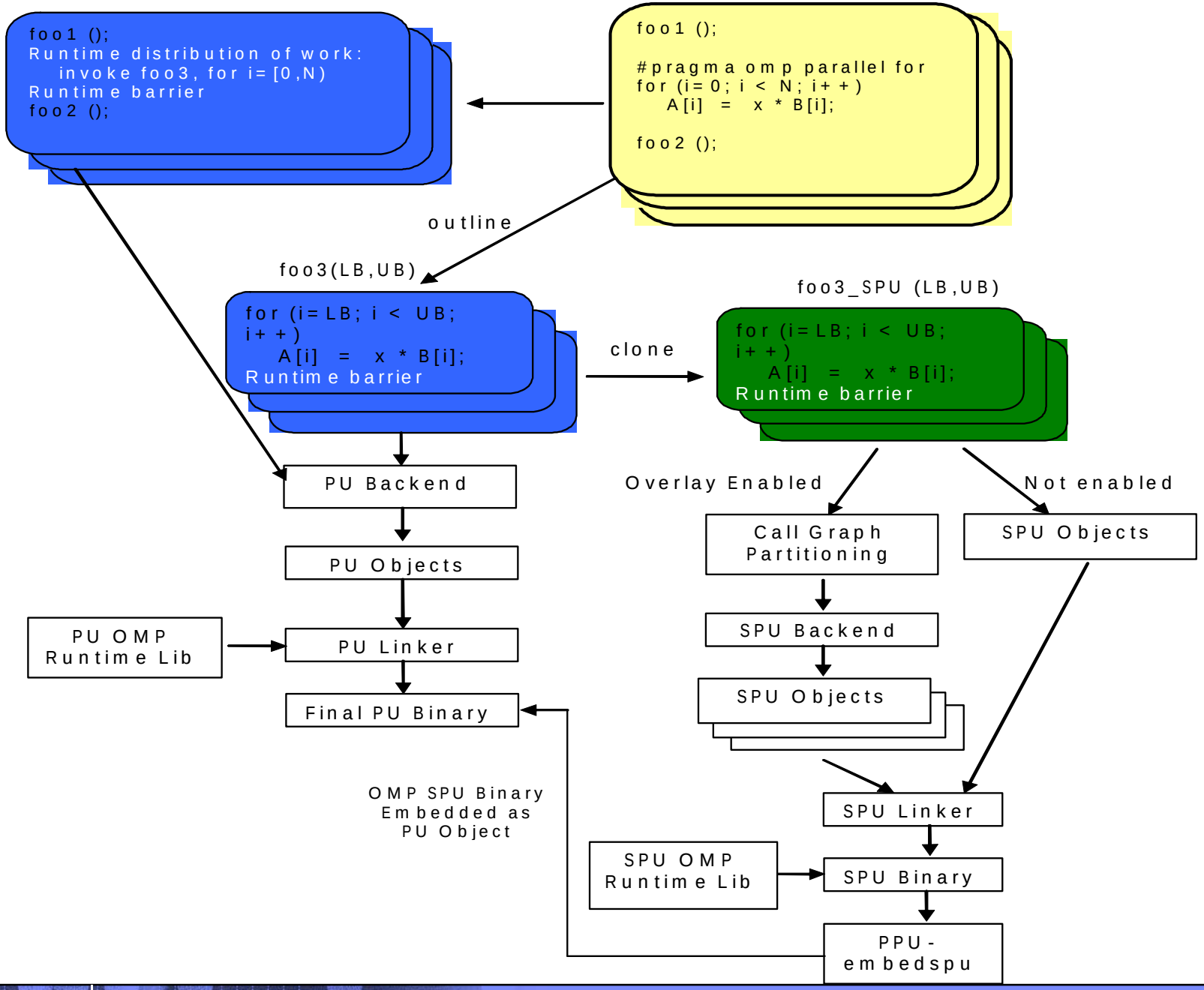
Code Generation – Code Overlay Support

- **When the size of the functions for SPEs is too big, code cannot fit into local storage**
  - Lots of parallel loops
- **Call-graph partitioning and code overlay support (in SDK)**
  - Partition the sub-graph of SPE functions into several partitions
  - Each call graph partition generates one SPE object file
  - Create a code overlay for each SPE object file
  - Code overlays share code address space
  - Load into local memory on demand
- **Code size normally not an issue**
  - Except for huge single function

Code Generation – Example



Code Generation – Summary



Outline

- **Compiler Overview**
- **Compiler in detail**
  - Code Generation
  - Thread and Synchronization
  - Data Management
- **Conclusion**

Threads and Synchronization – Master Thread

- **OpenMP threads execute on both PPE and SPEs**
  - PPE can choose not to participate in work sharing
- **PPE thread is the master thread**
  - PPE is designed to manage SPEs
  - Smaller and simpler code for SPE runtime (in limited SPE local store)
- **Responsibilities**
  - Creating SPE threads
  - Distributing and scheduling tasks
  - Initiating synchronization operations
  - Handling all OS service requests

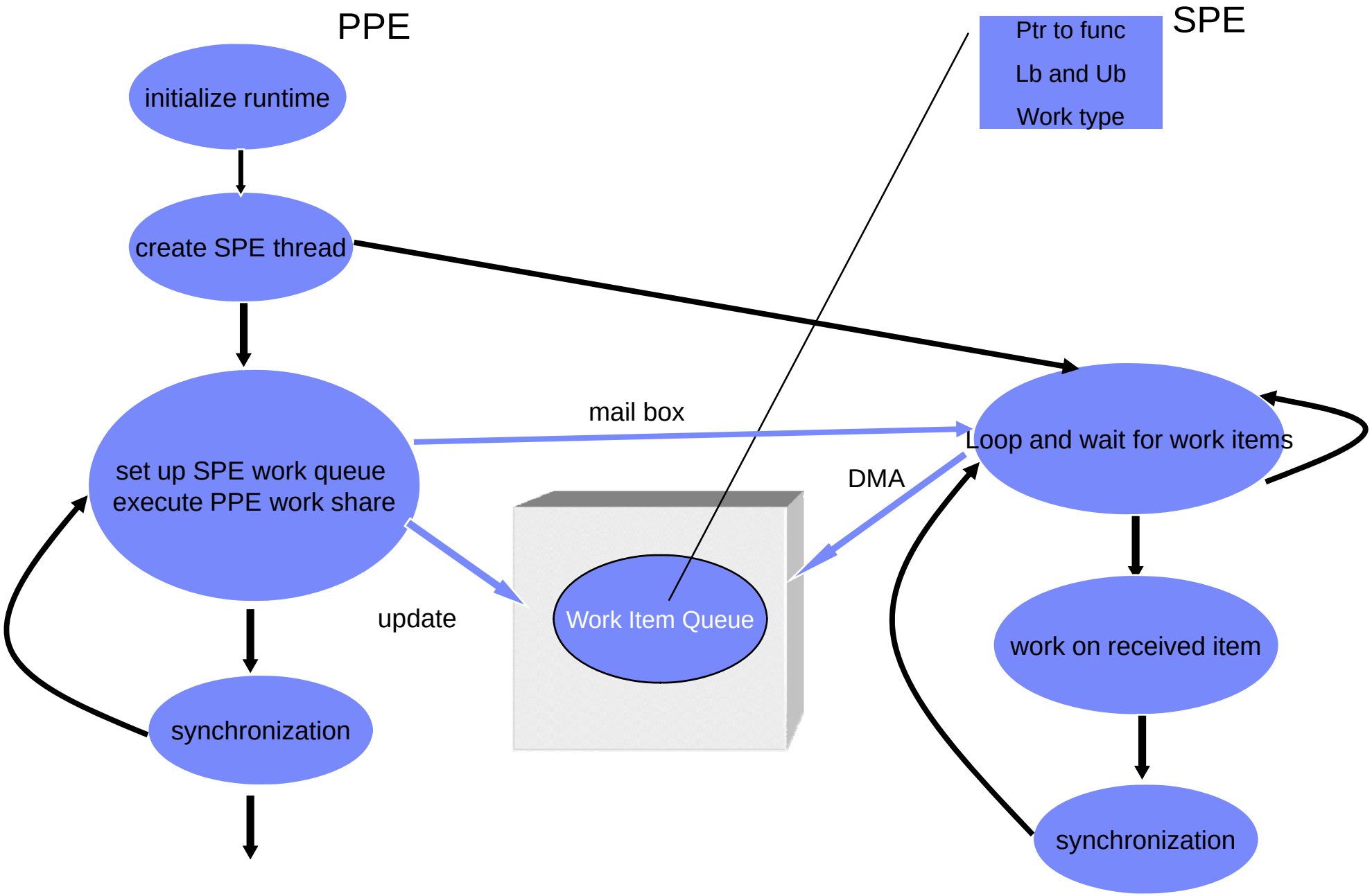
Threads and Synchronization – Tasks

- **Some kind of work to be done by OpenMP threads**
  - Execution of an outlined parallel region, loop, or section
  - Cache flush
  - Barrier synchronization
- **Task queue**
  - A task queue for each SPE thread
  - Resides in system memory and contains all the tasks to be executed by the thread
  - PPE thread creates tasks and puts tasks into the task queue
    - Task type, lower bound and upper bound for a parallel loop, function pointer and etc.
  - SPE threads fetch and execute tasks, and update task queue (using DMA)

Threads and Synchronization – Synchronization

- **Follow OpenMP specification**
  - Barriers, locks, cache flushes
- **Implementation**
  - Mail boxes in Memory Flow Controller (MFC) for each SPE
    - For efficient communication of 32-bit values between cores
    - Used by PPE thread to indicate a SPE thread that tasks are available and exactly how many of them are available
    - Used in barrier implementation (Thanks to Dan Brokenshire)
  - Atomic unit in MFC for each SPE
    - Implementing atomic DMA commands
    - Used for efficient implementation of OpenMP locks and cache flush operations

Execution Flow



Outline

- **Compiler Overview**
- **Compiler in detail**
  - Code Generation
  - Thread and Synchronization
  - Data Management
- **Conclusion**

Data Management

- **Memory model: Relaxed consistency, shared memory model**
  - Each thread can have its own temporary view of memory
  - Until forced to share memory by an OpenMP flush operation
- **Private data allocated in SPE local store**
  - Easy access
- **Shared data resides in system memory**
  - Not directly accessible by SPEs
  - Global variable addressed through **CESOF support**
  - Accessed through compiler-managed DMA operations
    - All data accesses can be handled by **software data cache**
    - Regular data accesses can be optimized with **direct buffering**

Software Data Cache

- **Works just like a hardware cache, but implemented in software**
  - Loads/stores replaced with software cache lookup instructions
  - Miss handler invoked for a cache miss
    - Brings in the missing cache line
    - Evicts an existing cache line if necessary
  - Currently fixed configuration (64KB,128B cache line, 4-way) for efficient implementation – will be made configurable later
- **Coherence among threads**
  - One cache line may be shared by multiple SPE threads – cannot naively evict whole cache line
  - Dirty bits to record modified data (in unit of byte)
  - Atomic updates based on dirty bits to evict a cache line
- **Pros/Cons**
  - Uniform solution for all kinds of references
  - Exploit data reuse dynamically
  - Overhead

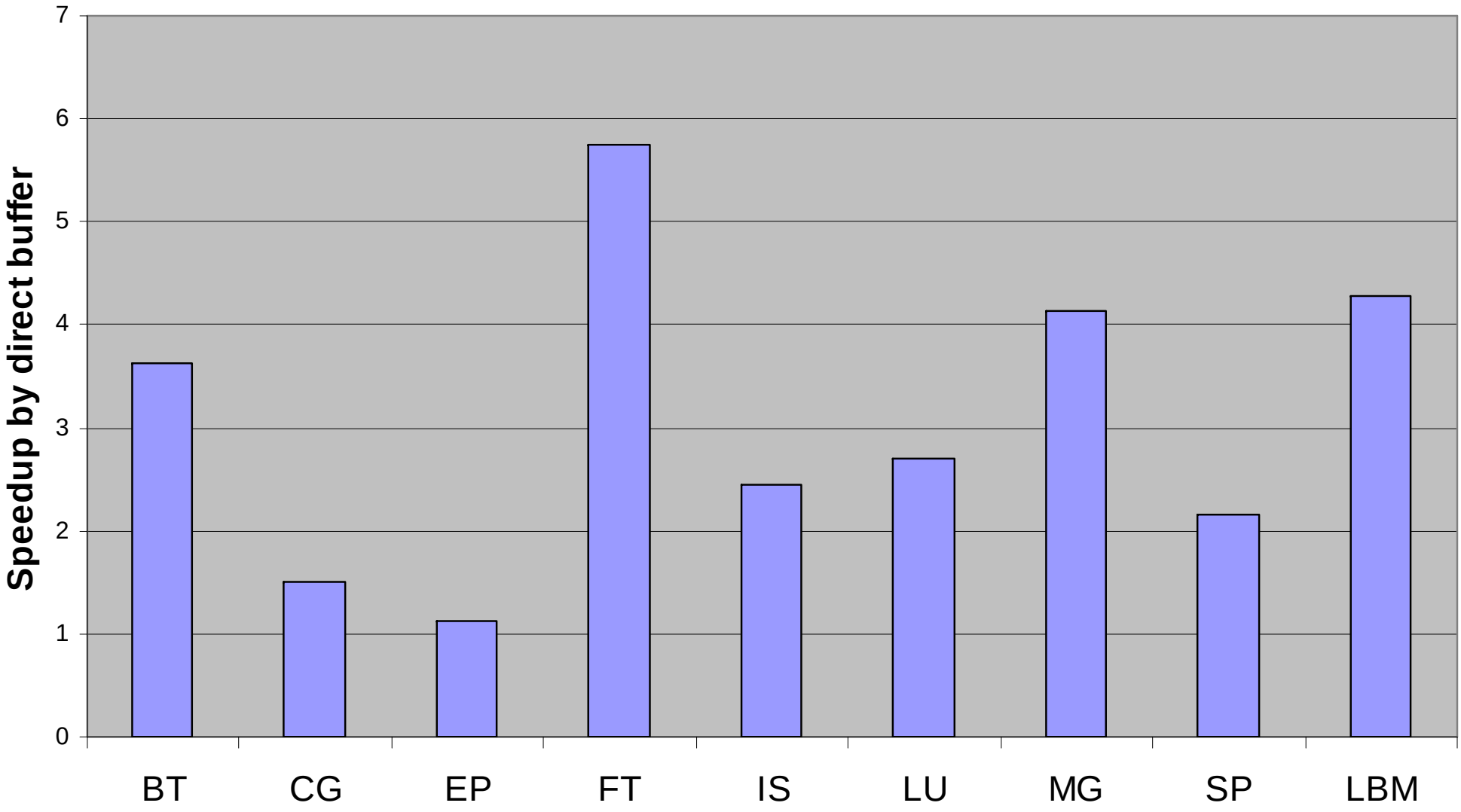
Direct buffering

- **Handles regular data accesses to shared memory by compiler**
  - Buffers in SPE local memory are controlled by compiler
  - Calls to allocate and free buffers are inserted
  - DMA operations are inserted
  - References to global variables are replaced by direct references to the local buffer
- **Pros/Cons**
  - Much less overhead: no lookup, more control on DMA
  - Decisions have to be made at compile time
    - Sometimes not possible
    - Sometimes not optimal

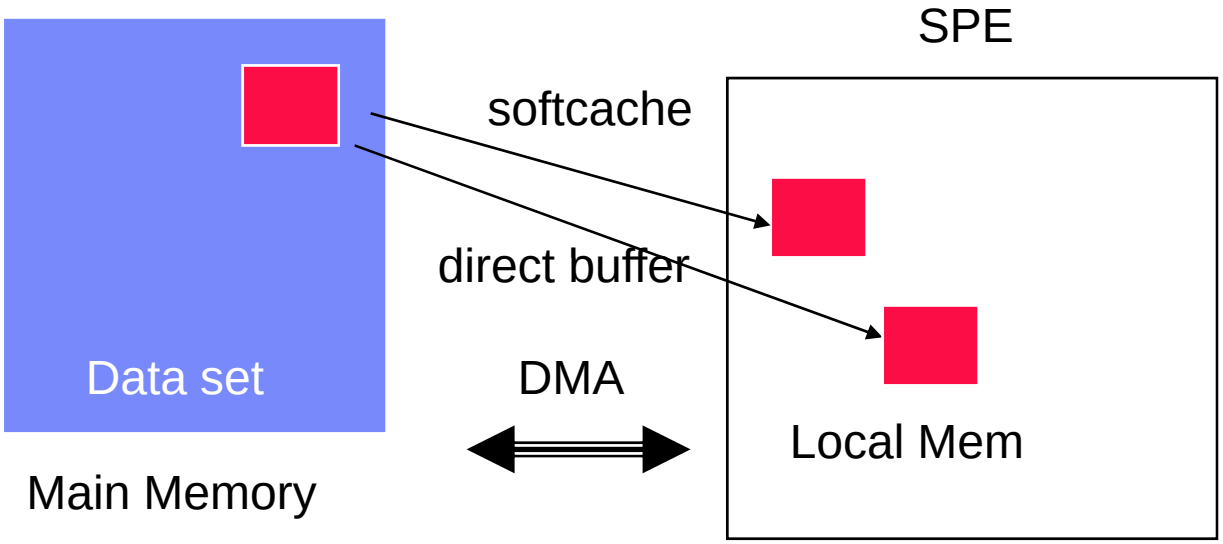
```
for (i=0; i<N; i++) {
    A[i] = B[i]*C[i]
}
```

```
for (ii=0; i<N; i+=bf) {
        read part B into B';
        read part C into C';
        for (i=ii; i < min(ii+bf, N); i++) {
                A'[i]=B'[i]*C'[i];
        }
        write A' back to A;
}
```

Software Cache vs. Direct Buffering

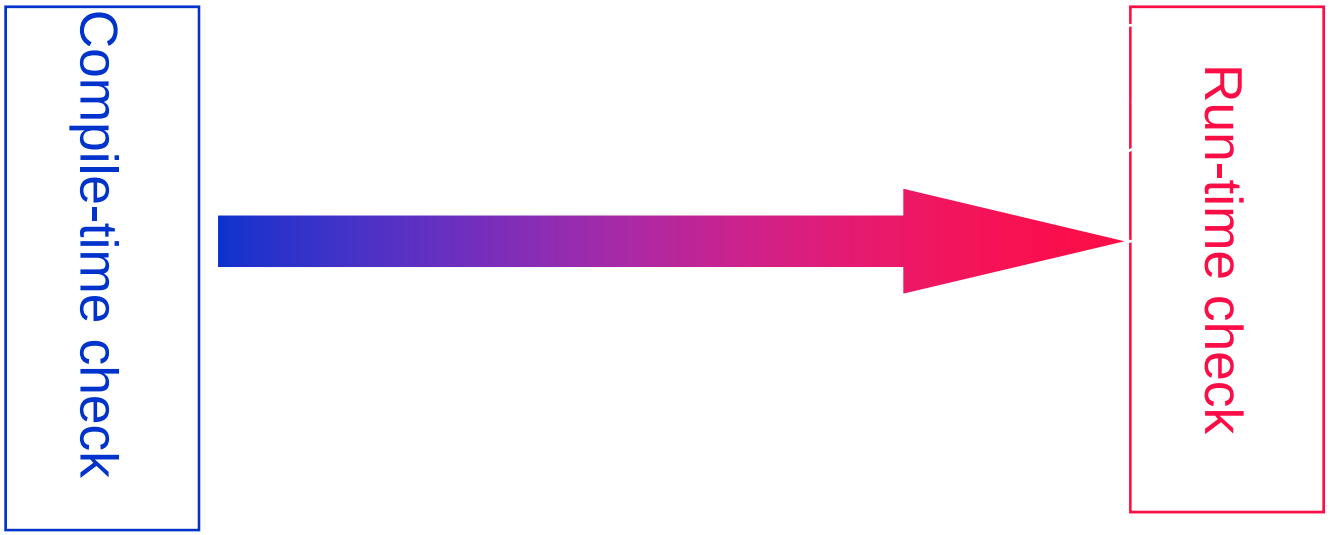


Integrating Data Management Techniques



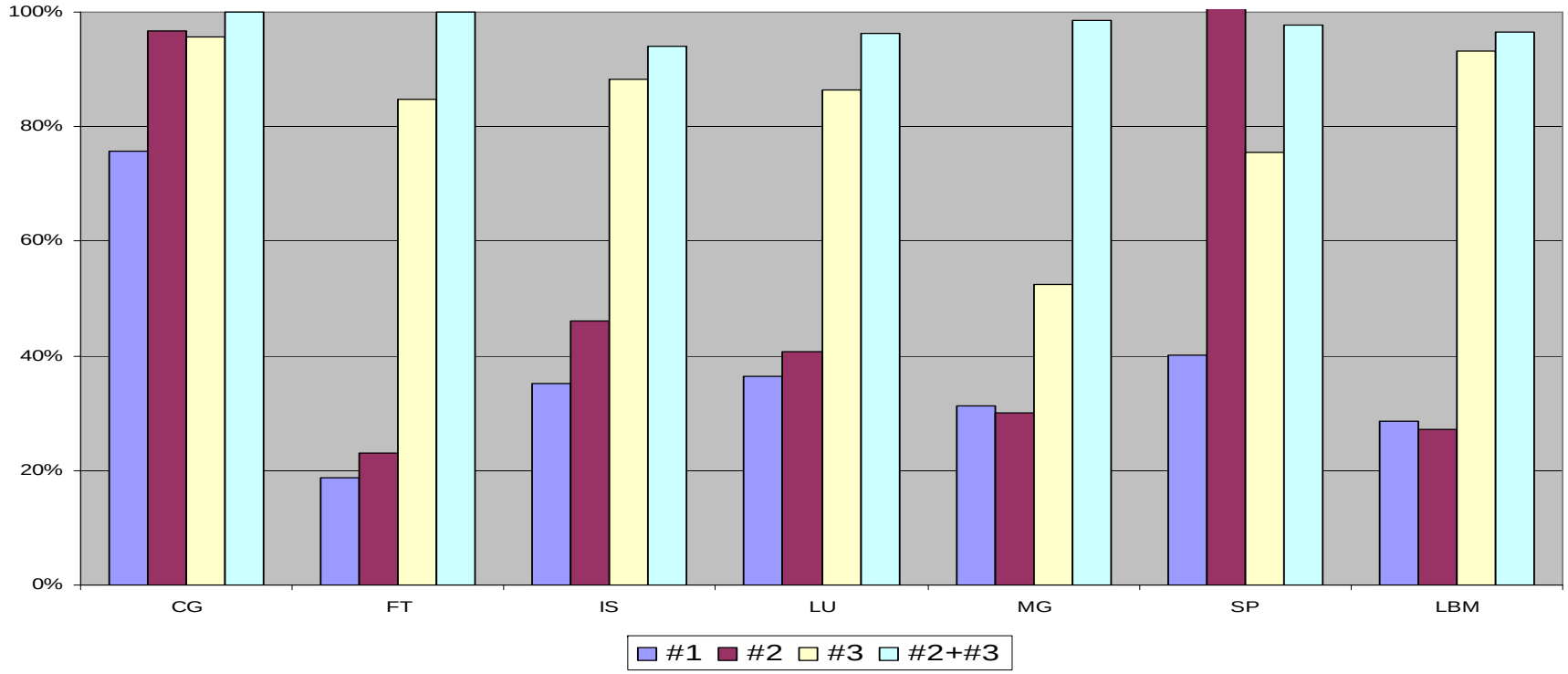
- **What if multiple copies of data**
  - Software controlled cache and direct buffer
  - Multiple direct buffers
- **Incorrect result may be produced!**

Solution Exploration



- **Explore both compile-time and run-time check**
  - Compile-time method efficient
  - Run-time method powerful
- **Goal: better performance**
- **Refer to ICS08 paper and more coming**

Performance of Different Approaches

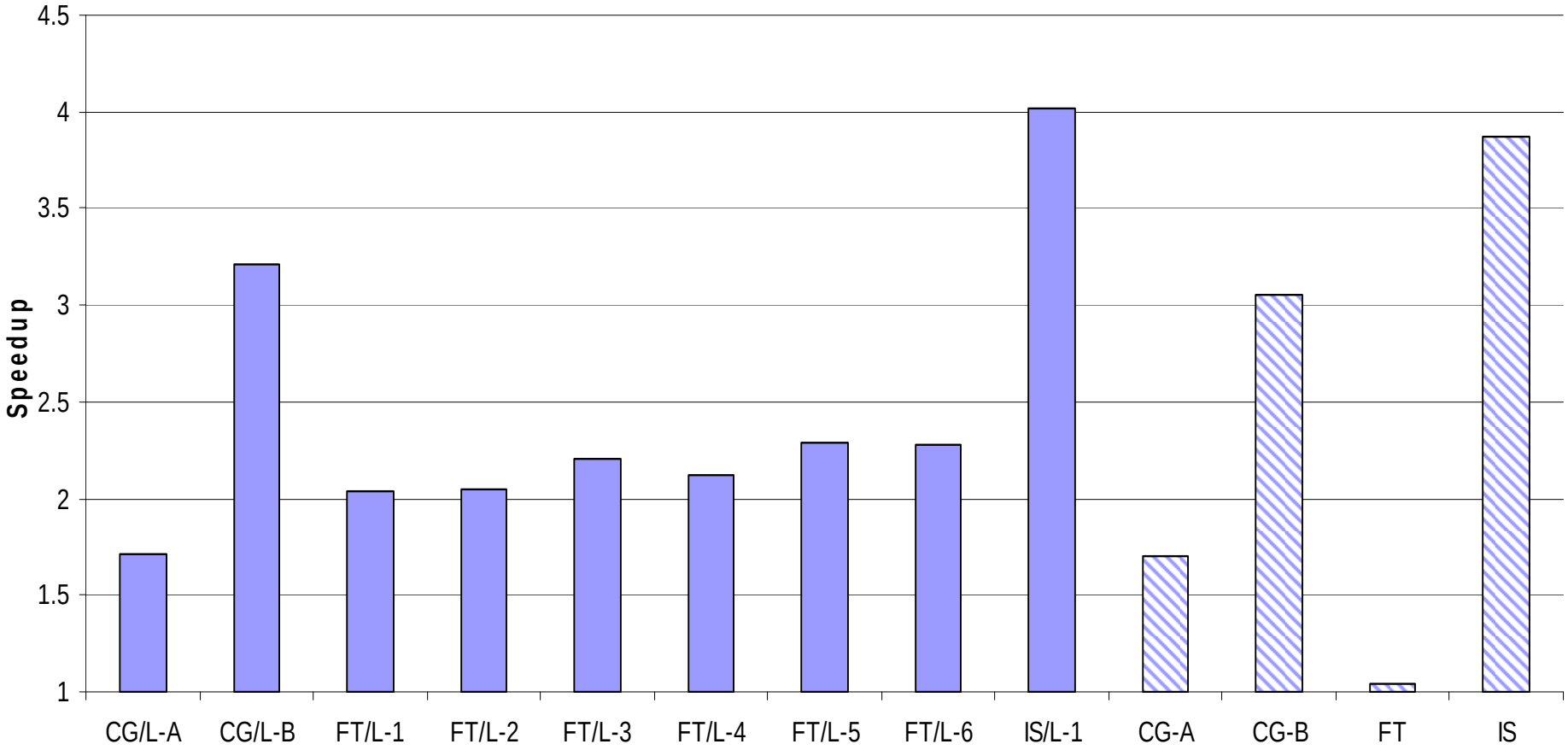


- Performance normalized to no coherence maintenance overhead
- #1 pure compiler approach
- #2 compiler analysis and runtime boundary coherence maintenance
- #3 full runtime coherence maintenance

Prefetching for Software Cache

- **The cache miss is expensive (more than 1000 cycle)**
  - Blocked DMA
  - Context switching for jumping to miss handler
  - Cache maintenance
- **Prefetching?**
  - Not hardware supported prefetch instruction
  - Rely on DMA only
  - How to sync the use of cache and prefetching on the fly
- **Our solution**
  - Inspector-consumer model
  - Feasible and efficient for DMA
  - Special optimizations
- **Refer to CGO'08 paper**

Speedup for Loops by Prefetching



Conclusions

- **Supporting OpenMP on Cell facilitates code reuse and new application development**
- **Our accomplishments**
  - For simple test cases, our OpenMP compiler achieves performance similar to hand-optimized implementation
  - For larger benchmarks, some of them show significant performance gains
  - Feasible to extract high performance on Cell using easy-to-use OpenMP programming model
- **Future work**
  - Optimize implementation
  - Improve performance on a wider set of applications

For More Information

Visit our website:

www.research.ibm.com/cellcompiler/compiler.htm

Paper:

- Tong Chen, Haibo Lin, Tao Zhang, Kathryn O'Brien, Kevin O'Brien, "Orchestrating Data Transfer for the Cell/B.E." accepted by International Conference on Supercomputer (ICS) 2008
- Tong Chen, Tao Zhang, Zehra Sura, Marc Gonzalez Tallada, Kathryn O'Brien, Kevin O'Brien, , "Prefetch Irregular References for Software Cache on Cell", International Symposium on Code Generation and Optimization (CGO) 2008.
- Kevin O'Brien, Kathryn O'Brien, Zehra Sura, Tong Chen and Tao Zhang, "Supporting OpenMP on Cell", International Workshop on OpenMP (IWOMP), 2007
- Tong Chen, Zehra Sura, Kathryn M. O'Brien, John K. O'Brien: "Optimizing the Use of Static Buffers for DMA on a CELL Chip" Workshops on Language and Compiler for Parallel Computation (LCPC) 2006: 314-329
- Alexandre E. Eichenberger, Kathryn M. O'Brien, Kevin O'Brien, Peng Wu, Tong Chen, Peter H. Oden, Daniel A. Prener, Janice C. Shepherd, Byoungro So, Zehra Sura, Amy Wang, Tao Zhang, Peng Zhao, Michael Gschwind: Optimizing Compiler for the CELL Processor. IEEE PACT 2005: 161-172
- Alexandre E. Eichenberger, Peng Wu, Kevin O'Brien: Vectorization for SIMD architectures with alignment constraints. PLDI 2004: 82-93

Backup