MERCURY
COMPUTER SYSTEMS

A Multi-Paradigm Programming Model for Heterogeneous Architectures

Michael Champigny
Research Scientist
Advanced Computing Solutions
Mercury Computer Systems

2009 HPEC Workshop
September 23, 2009

Copyright © 2009 Mercury Computer Systems

Parallelism drives performance scaling

MERCURY COMPUTER SYSTEMS

- But how do we expose parallelism...
  - on embedded platforms?
  - across heterogeneous devices?
  - using proven programming models?
  - with any C++ toolchain?
  - without rewriting code?
  - without introducing deadlock?
  - without losing compiler optimizations?

Copyright © 2009 Mercury Computer Systems

Embedded requires a unique solution

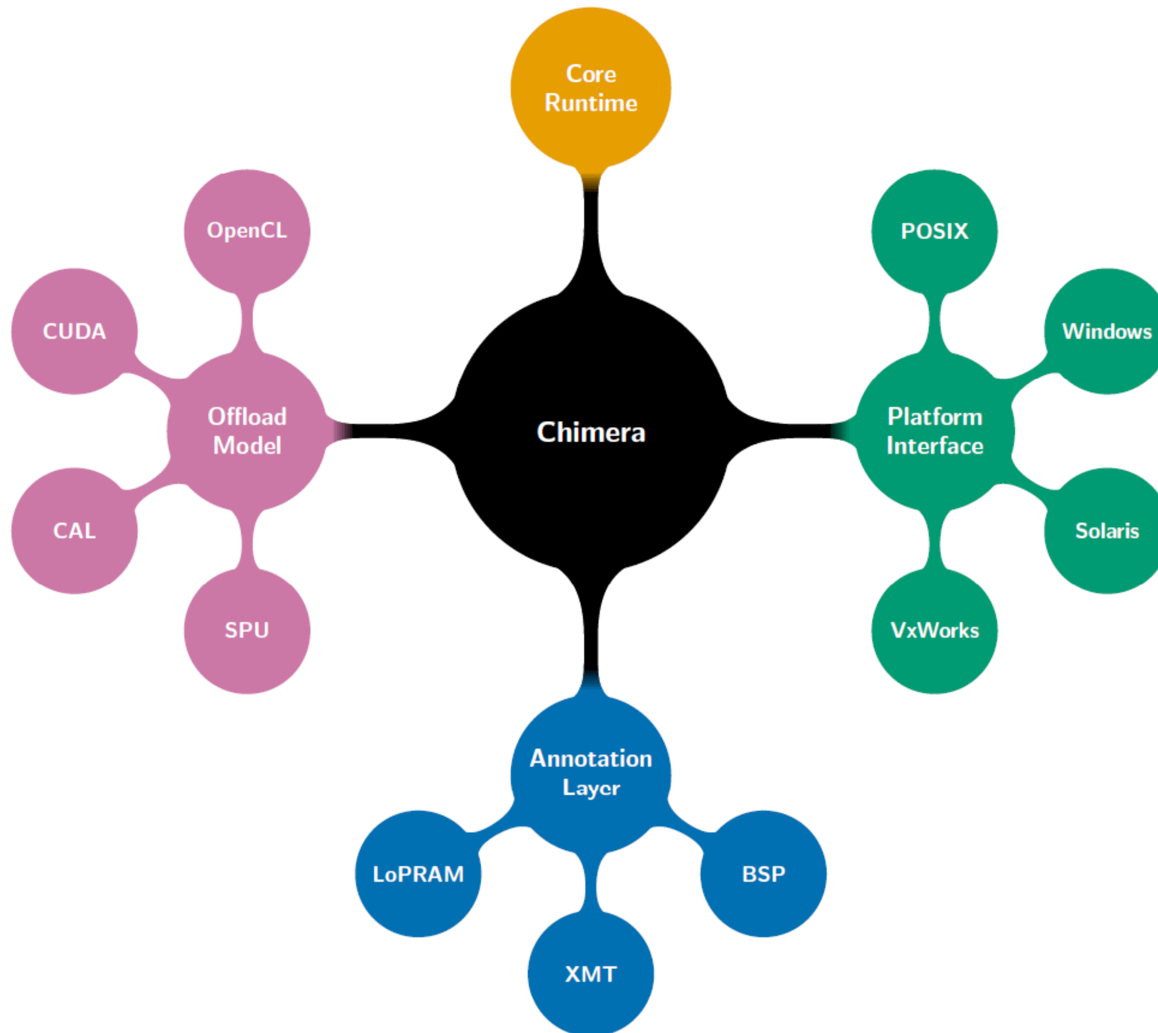
MERCURY COMPUTER SYSTEMS

Solution	Any Compiler	Offload	Nested Tasks	Pipelines	Language
OpenMP	✗	✗	✓	✓	✓
OpenCL	✓	✓	✗	✓	✗
Cilk++	✗	✗	✓	✗	✓
TBB	✓	✗	✓	✓	✗

- Broad portability across toolchains
- Lightweight and high performance
- Compiler-agnostic
- Multi-paradigm
- Heterogeneous

Copyright © 2009 Mercury Computer Systems

Chimera: a C++ concurrency platform



Dynamic runtime scheduler

MERCURY COMPUTER SYSTEMS

- Coherent shared memory platforms
- Load balancing across devices
- Greedy work-stealing algorithm
- Heuristics improve cache hit ratio
- Scalable memory management

Copyright © 2009 Mercury Computer Systems

Extend ISO C++ to expose parallelism

- Support parallel programming models
  - LoPRAM: Low Degree Parallel RAM
  - XMT: eXplicit Multi-Threading
- Compiler independence through macros
- Leverage the standard C preprocessor
- Enable reuse of published algorithms
- Decouple client from execution model

Copyright © 2009 Mercury Computer Systems

LoPRAM parallel programming model

- LoPRAM programming model
  - Low Degree Parallel RAM
  - Assumes $O(\log n)$ processors available
  - Parallelism exposed as tasks
  - Popular MIMD model for CMP
  - CREW memory model

```
VECTOR-ADD(A, B, C, n)

1  if n < 64
2     then VECTOR-ADD-SERIAL(A, B, C, n)
3     else q ← n/2
4           ▷ pal-threads
5             VECTOR-ADD(A, B, C, q)
6             VECTOR-ADD(A + q, B + q, C + q, n − q)
7           ▷ nowait
```

Copyright © 2009 Mercury Computer Systems

XMT parallel programming model

- XMT programming model
  - eXplicit Multi-Threading
  - Assumes $O(n)$ processors available
  - Parallelism exposed as parallel loops
  - Popular SIMD model for VU/GPU
  - CRCW memory model

$\textsc{Vector-Add}(A, B, C, n)$

1 $\triangleright$ spawn(n)
2 $\quad C[\$] \leftarrow A[\$] + B[\$]$

Copyright © 2009 Mercury Computer Systems

Example of Chimera LoPRAM annotations

```
1  void vadd parallel (
2    float* A, float* B, float* C, int n)
3  {
4    if (n < 64)
5      vadd_serial (A, B, C, n);
6    else {
7      const int q (n/2);
8      fork_nowait (vadd, A, B, C, q);
9      vadd serial (A+q, B+q, C+q, n-q);
10   }
11 }
```

Copyright © 2009 Mercury Computer Systems

Example of Chimera XMT annotations

```
1  void vadd parallel (
2    float* A, float* B, float* C, int n)
3  {
4    fork_forall (n) {
5      C[$] = A[$] + B[$];
6    }
7  }
```

Example of Fibonacci sequence generation

MERCURY COMPUTER SYSTEMS

- **Fibonacci sequence as a recurrence**

fibonacci

$$F_n = \begin{cases} 0 & \text{if } n = 0, \\ 1 & \text{if } n = 1, \\ F_{n-1} + F_{n-2} & \text{if } n > 1. \end{cases}$$

FIBONACCI(n)

```
1  if n < 20
2     then return FIBONACCI-SERIAL(n)
3     else ▷ pal-threads
4             x ← FIBONACCI(n − 1)
5             y ← FIBONACCI(n − 2)
6  return x + y
```

Copyright © 2009 Mercury Computer Systems

Example Fibonacci code in Cilk++

```
1  int fib (int n)
2  {
3    if (n < 20)
4      return fib_serial (n);

6    int x;
7    x = cilk_spawn fib (n-1);
8    int y = fib (n-2);
9    cilk_sync;

11   return x + y;
12 }
```

Example Fibonacci code in OpenMP

```
1   int fib (int n)
2   {
3     if (n < 20)
4       return fib_serial (n);

6     int x, y;
7     #pragma omp task shared(x)
8     x = fib (n-1);
9     #pragma omp task shared(y)
10    y = fib (n-2);
11    #pragma omp taskwait

13    return x + y;
14  }
```

Example Fibonacci code in Chimera using `future`

MERCURY COMPUTER SYSTEMS

```
1   int fib parallel (int n)
2   {
3     if (n < 20)
4       return fib_serial (n);

6     future(int) x = fork_future (fib, n-1);
7     int y = fib serial (n-2);

9     return x + y;
10  }
```

Example Fibonacci code in Chimera using join

```
1  int fib parallel (int* sum, int n)
2  {
3    if (n < 20) {
4      *sum = fib_serial (n);
5      return;
6    }

8    int x, y;
9    join {
10     fork (fib, &x, n-1);
11     fib serial (&y, n-2);
12   }
13   *sum = x + y;
14 }
```

Copyright © 2009 Mercury Computer Systems

Preserving serial semantics in source code

- Parallel to serial transformation rules
- Define `CHIMERA_SERIAL` to apply rules

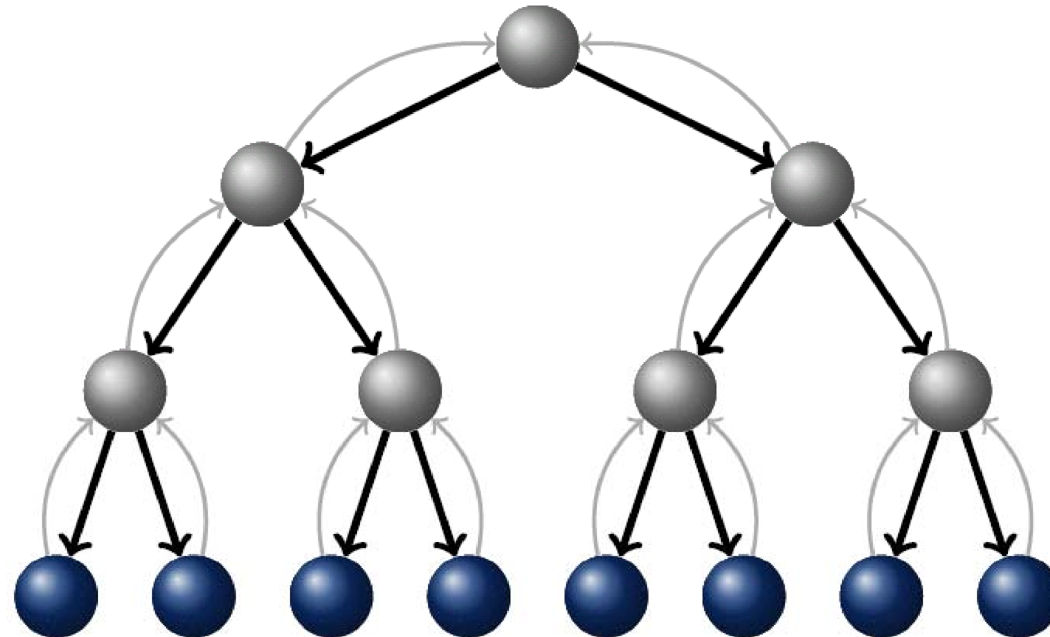
parallel $\longmapsto$ *serial*

$$\begin{aligned}
\text{join} &\longmapsto \lambda \\
\text{serial} &\longmapsto \lambda \\
\text{parallel} &\longmapsto \lambda \\
\text{atomic} &\longmapsto \lambda \\
\text{future}(\tau) &\longmapsto \tau \\
\text{fork}\ (f, a_1, \ldots, a_n) &\longmapsto f\ (a_1, \ldots, a_n)
\end{aligned}$$

Copyright © 2009 Mercury Computer Systems

Fully-strict computation DAG

MERCURY COMPUTER SYSTEMS

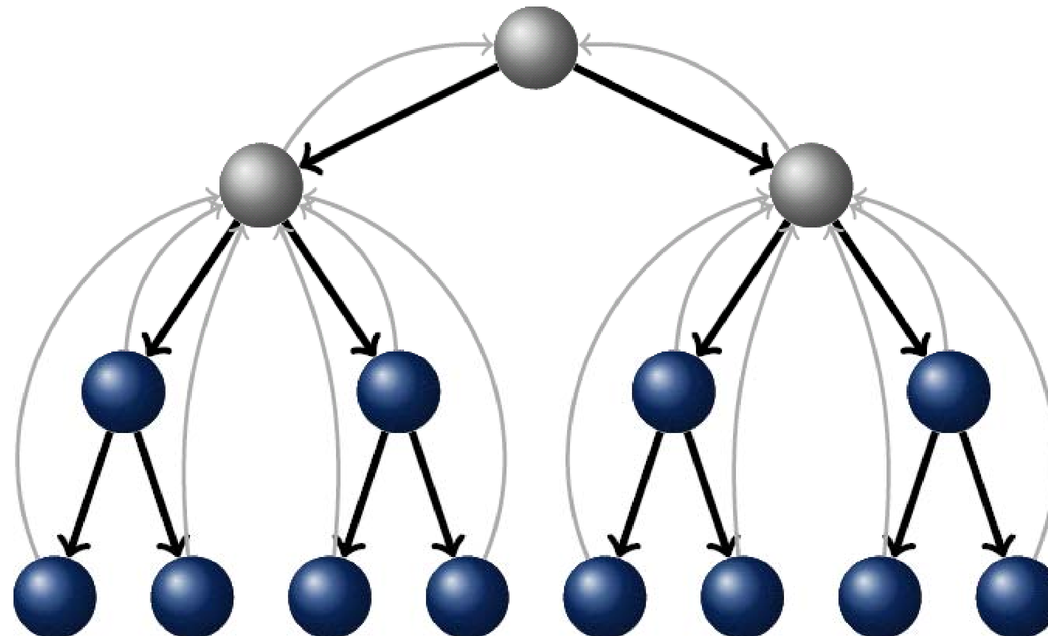


- Parents must wait for children to finish
- Can impose unnecessary synchronization
- Cilk++ is restricted to fully strict DAG

Copyright © 2009 Mercury Computer Systems

Terminally-strict computation DAG

MERCURY
COMPUTER SYSTEMS



- Parents may finish before children
- Relaxed ordering reduces synchronization
- Chimera supports terminally strict DAG

Copyright © 2009 Mercury Computer Systems

Speedup on numerical microbenchmarks

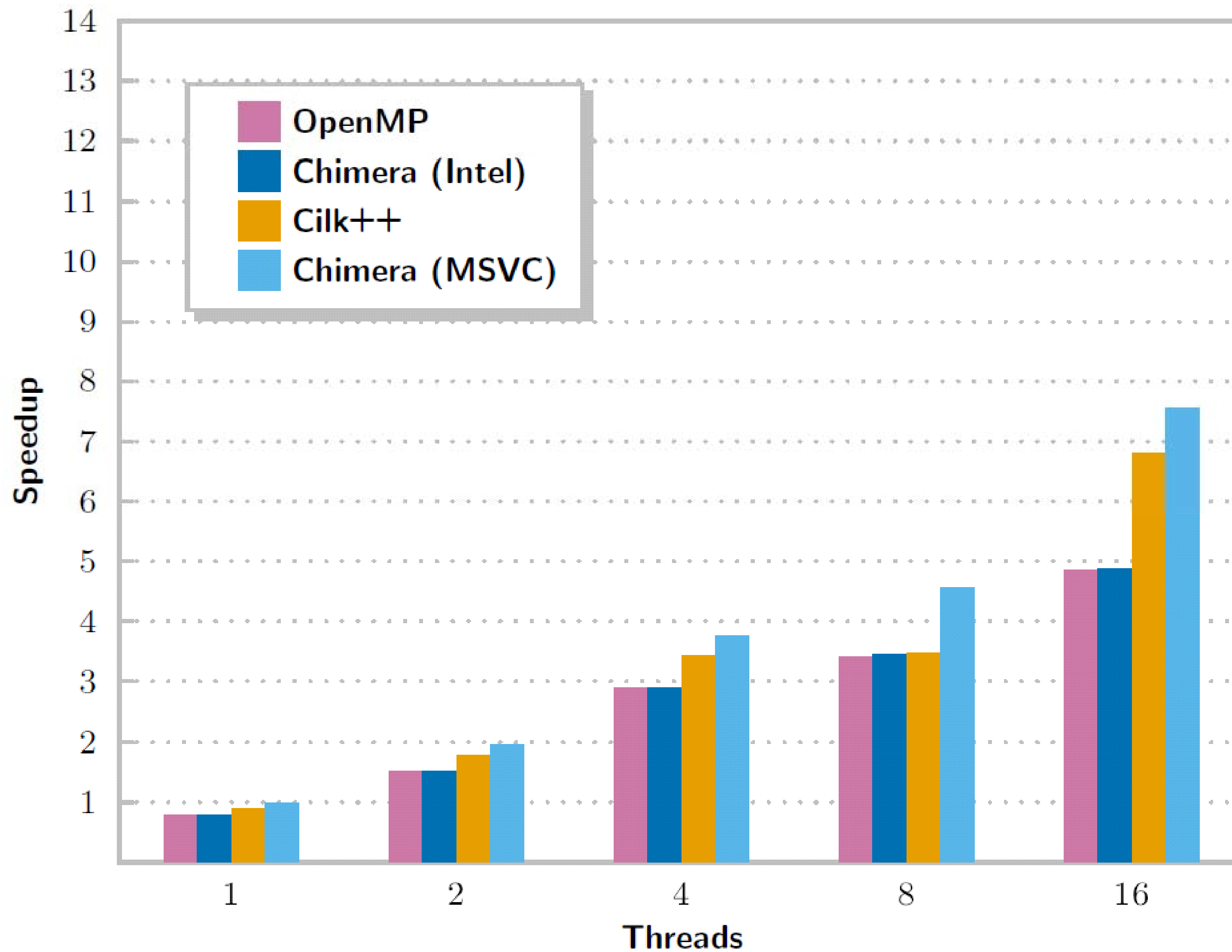
MERCURY COMPUTER SYSTEMS

- Adapted from MIT Cilk examples
- Baseline is the serial version
- Measured best out of 50 runs
- Windows Server 2008 SP2
- 2P Intel 2.67GHz Xeon Quad Server
- OpenMP 3.0 using Intel 11.1 32-bit
- Cilk++ 1.0.3 using MSVC 9 32-bit

Copyright © 2009 Mercury Computer Systems

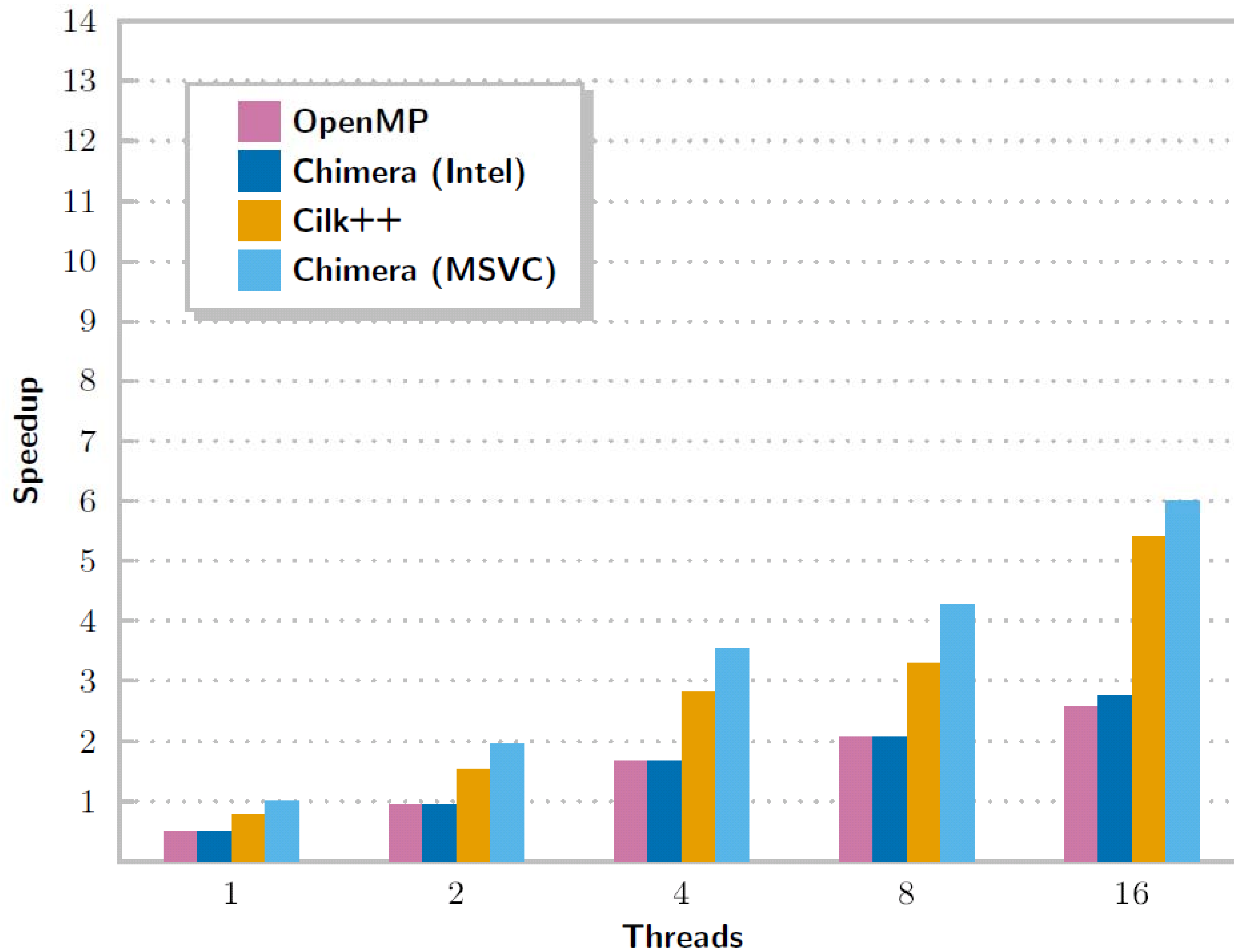
Matrix-multiply speedup

MERCURY COMPUTER SYSTEMS



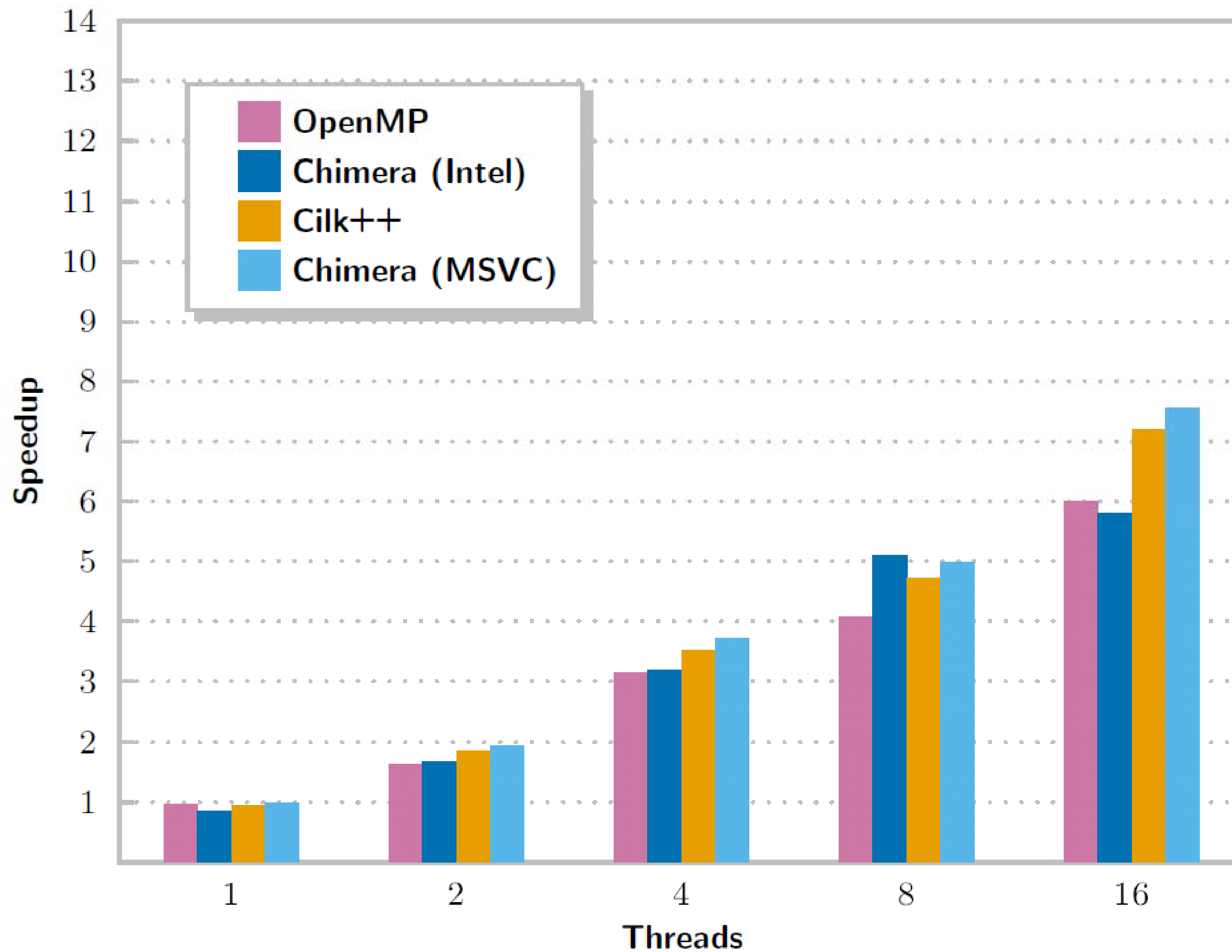
Copyright © 2009 Mercury Computer Systems

Heat diffusion speedup



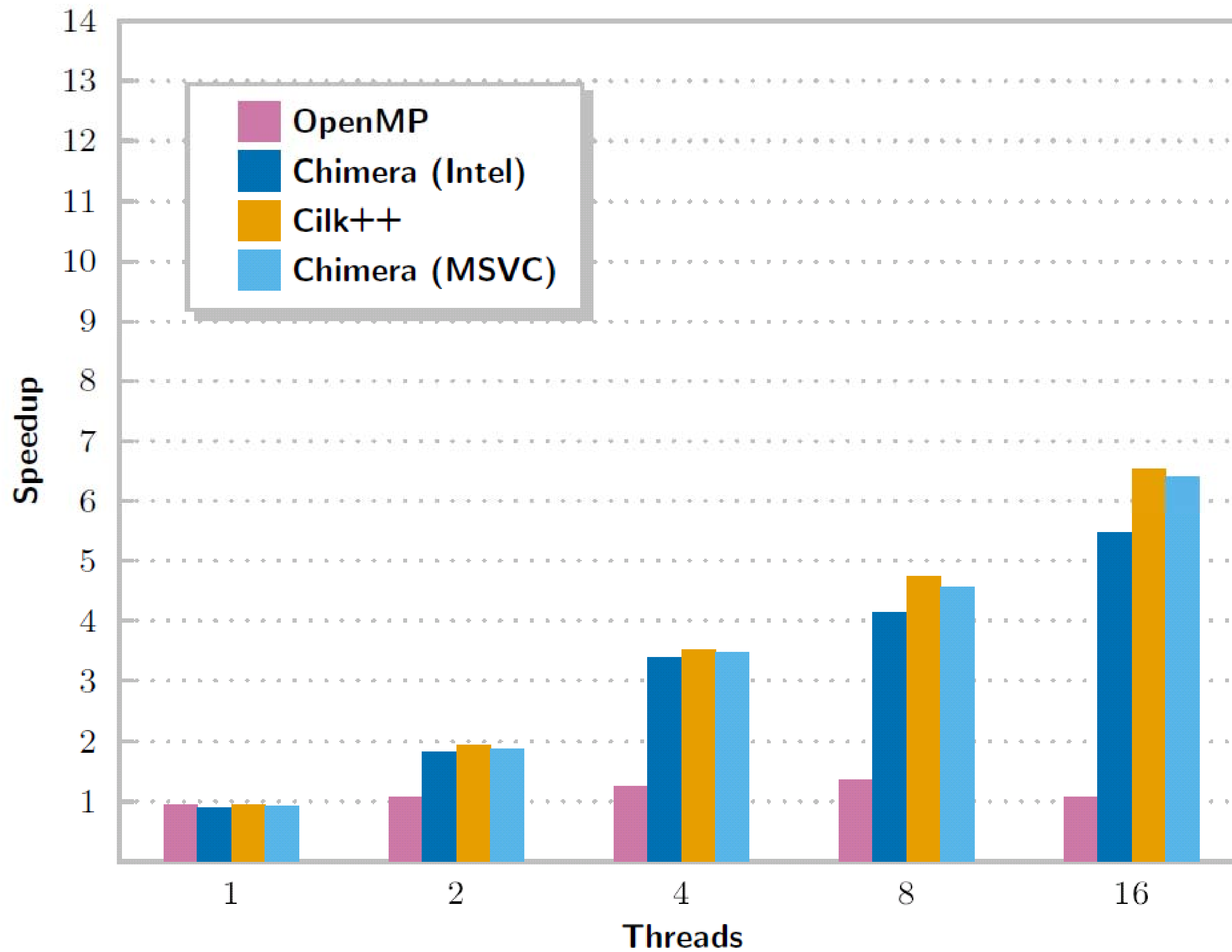
Copyright © 2009 Mercury Computer Systems

LU-decomposition speedup



Copyright © 2009 Mercury Computer Systems

FFT speedup



Copyright © 2009 Mercury Computer Systems

High productivity without a new language or compiler

MERCURY COMPUTER SYSTEMS

- Latency comparable to industry solutions
- Exploit performance of vendor compilers
- Competitive parallel efficiency $\frac{T_p}{P}$ at scale
- Low overhead on a single core
- Productivity is now a first-order concern...
- Let me know what you think!
- mchampig@mc.com

Copyright © 2009 Mercury Computer Systems