Comparison of Three Popular Parallel Programming Models on the Intel Xeon Phi

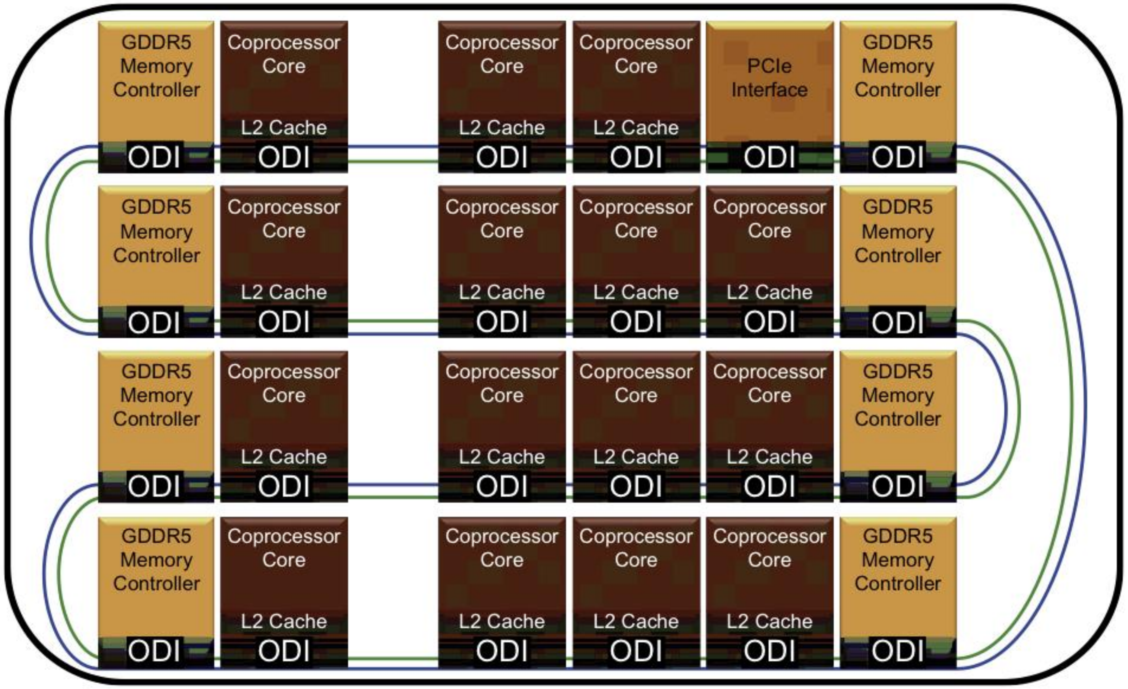
Ashkan Tousimojarad & Wim Vanderbauwhede

University of Glasgow sicsa*

Overview

- **Platform**
  - Intel Xeon Phi
- **Parallel Programming Models**
  - Intel OpenMP
  - Intel Cilk Plus
  - Intel Threading Building Blocks (TBB)
- **Solo Benchmarks**
  - Fibonacci
  - MergeSort
  - MatMul
    - ✓ Vectorization
- **Multiprogramming**
  - Running the benchmarks together!
- **The Bigger Picture**
- **Conclusion**

Platform: Intel Xeon Phi



- **L1I cache**: 32KB **L1D Cache**: 32KB
- **Architecture**: MIC
- **No of Cores**: 240 Logical Cores (60 Physical)
- 32 Vector Registers
- **L2 Cache**: 512KB
- **Shared Memory**: 8GB, 5GT/s
- **Core Frequency**: 1.053 GHz
- 512-bit wide VPU (16 single-precision)

University of Glasgow sicsa*

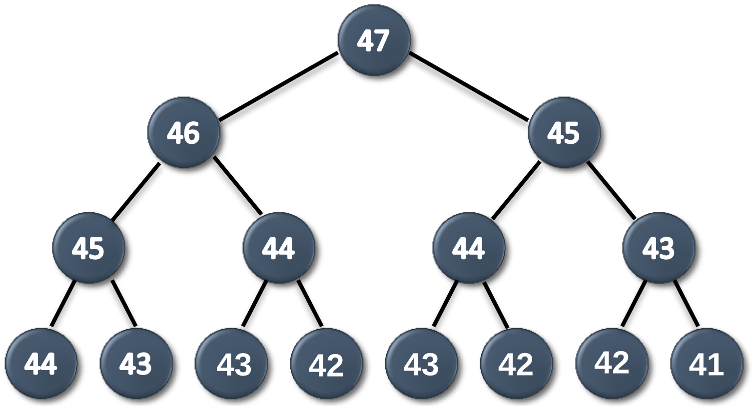
Parallel Programming Models

- **OpenMP**
  - An API using the fork-join model
  - Data parallelism – Task parallelism since OpenMP 3.0
  - Pragma
  - `#pragma omp parallel for`
- **Intel Cilk Plus**
  - An extension to C/C++
  - Simplicity and efficient work stealing
  - Keywords
  - `Cilk_for (int i=0; i<n; i++)`
- **Intel TBB**
  - An Object-Oriented C++ runtime library
  - Restructuring of programs to fit the TBB templates
  - Skeleton
  - `parallel_for(blocked_range<size_t>(0,N), Foo(a));`

University of Glasgow sicsa*

Benchmarks: Fibonacci

- Recursive, 47th Fibonacci number
- Heavyweight children, Lightweight parents
- Unbalanced children tasks
- 2^{tree_depth} children tasks
- Task parallelism
- Integer operations

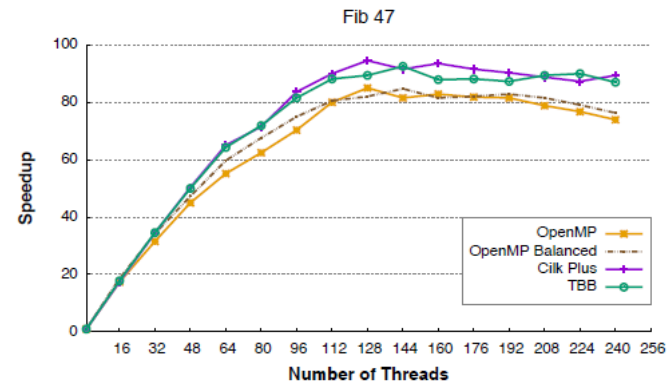


Total CPU Time

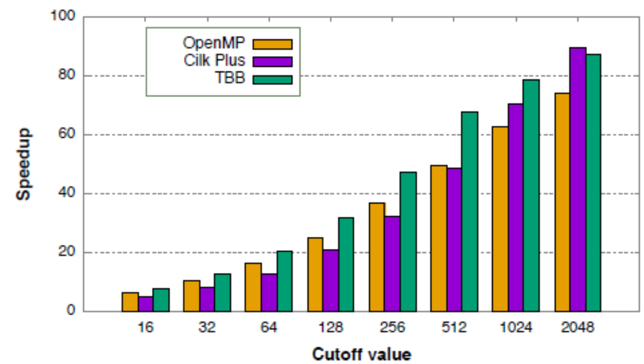
- Lower-better-metric
  - Measured using Intel VTune™ Amplifier XE 2013 performance analyzer
  - Plays an important role!

Benchmarks: Fibonacci

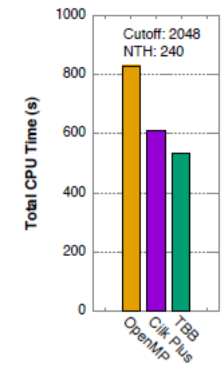
- **Speedup**
  - Cilk Plus and TBB are better
- **Cutoff**
  - Key to performance
- **Total CPU Time**
  - TBB consumes less
- **CPU Balance**
  - Does it necessarily imply an efficient load balance?



(a) Speedup, cutoff 2048, varying numbers of threads

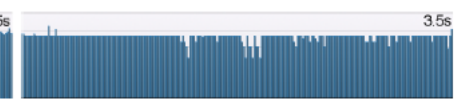


(b) Speedup, 240 threads, varying cutoffs

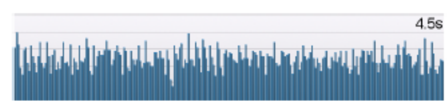


(c) Total CPU Time

(d) OpenMP, CPU balance



(e) Cilk Plus, CPU balance



(f) TBB, CPU balance

Benchmarks: MergeSort

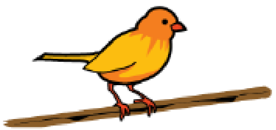
```
if (cutoff==1) SeqSort(A, tmp, size);
else {
 #pragma omp task
  Sort(A, tmpA, half, cutoff/2);
 #pragma omp task
  Sort(B, tmpB, size-half, cutoff/2);
 #pragma omp taskwait
 #pragma omp task
  Merge(A, B, size, tmp);
 #pragma omp taskwait
}
```

```
if (cutoff==1) SeqSort(A, tmp, size);
else {
 _Cilk_spawn Sort(A, tmpA, half, cutoff/2);
 _Cilk_spawn Sort(B, tmpB, size-half, cutoff/2);
 _Cilk_sync;
 _Cilk_spaw Merge(A, B, size, tmp);
 _Cilk_sync;
}
```

University of Glasgow sicsa*

Benchmarks: MergeSort

```
if (cutoff==1) SeqSort(A,tmp,size);
else {
 Sort& a = *new(allocate_child()) Sort(A,tmpA,half,cutoff/2);
 Sort& b = *new(allocate_child()) Sort(B,tmpB,size-half,cutoff/2);
 set_ref_count(3);
 spawn(a)
 spawn_and_wait_for_all(b);
 Merge& c = *new(allocate_child()) Merge(A,B,size,tmp);
 set_ref_count(2);
 spawn_and_wait_for_all(c);
}
```

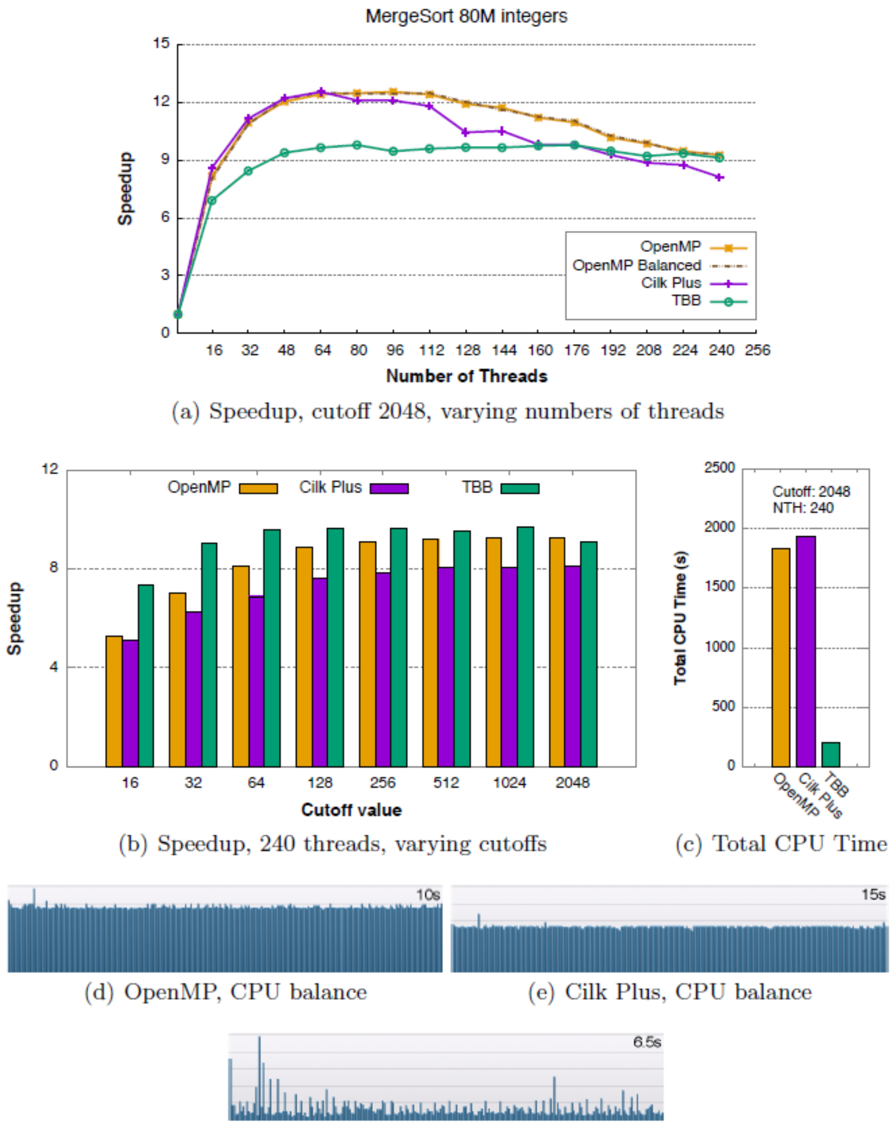


- Recursive: Merge Sort on an array of 80M integers
- Heavyweight children, Heavyweight parents
- Balanced children tasks
- Integer operations

University of Glasgow sicsa*

Benchmarks: MergeSort

- **Speedup**
  - Does not scale well (slowdown)
  - OpenMP and Cilk Plus can lead to better performance
- **Cutoff**
  - Larger than 64 is fine!
- **Total CPU Time**
  - TBB is significantly better
- **CPU Balance**
  - Does it necessarily imply an efficient load balance?
  - Parent tasks can run on the same core as one of their children



(a) Speedup, cutoff 2048, varying numbers of threads

(b) Speedup, 240 threads, varying cutoffs

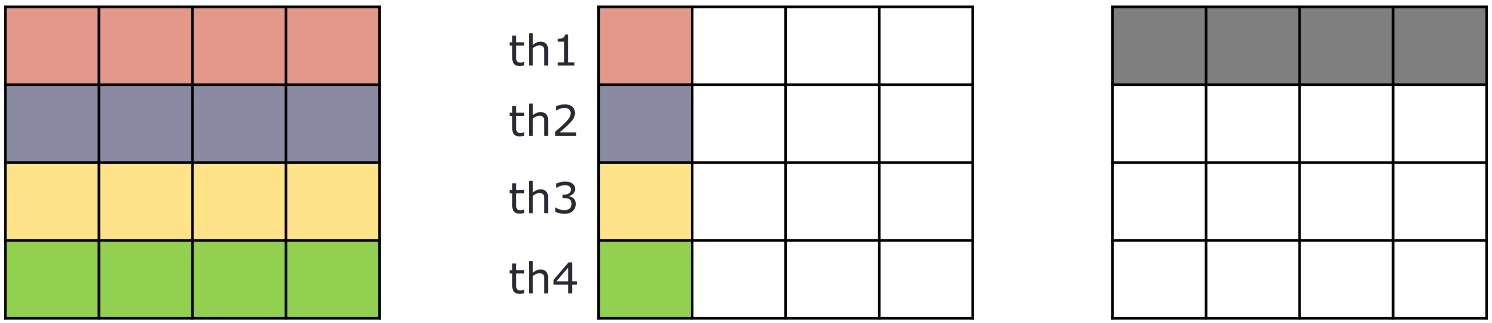
(c) Total CPU Time

(d) OpenMP, CPU balance

(e) Cilk Plus, CPU balance

(f) TBB, CPU balance

Benchmarks: MatMul



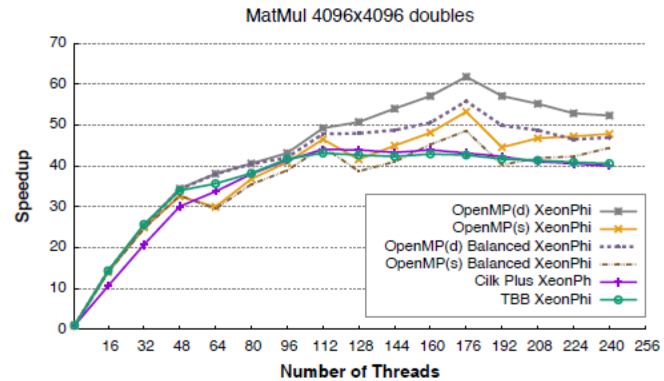
C[i][j] += **A**[i][k] * **B**[k][j]

- Matrix Multiplication of 4096x4096 double matrices (ikj)
- Vectorization
- Data parallelism
- Floating-point Operations

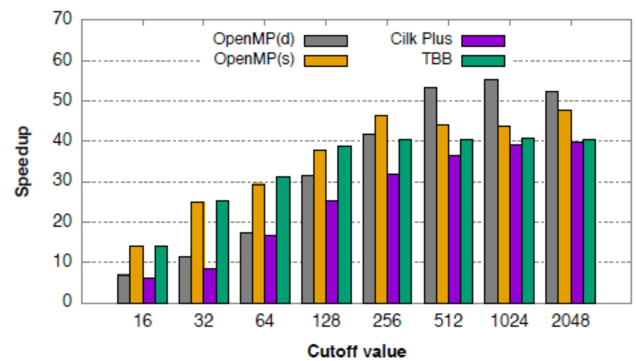
University of Glasgow sicsa*

Benchmarks: MatMul

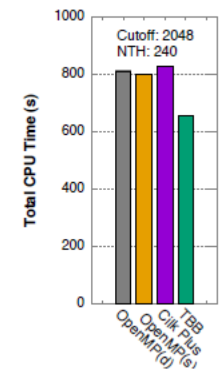
- **Speedup**
  - OpenMP(d) is the best
- **Cutoff**
  - 256 -> 512
- **Total CPU Time**
  - TBB consumes the least
- **CPU Balance**
  - Does it necessarily imply an efficient load balance?
  - Dynamic scheduling makes a visible change in the OpenMP case



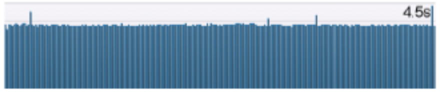
(a) Speedup, cutoff 2048, varying numbers of threads



(b) Speedup, 240 threads, varying cutoffs

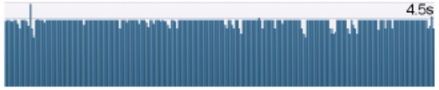


(c) Total CPU Time

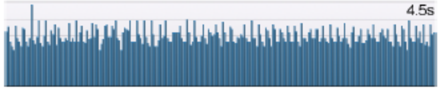


(d) OpenMP (dynamic), CPU balance

(e) OpenMP (static), CPU balance



(f) Cilk Plus, CPU balance



(g) TBB, CPU balance

University of Glasgow sicsa*

Overhead of Runtime Systems

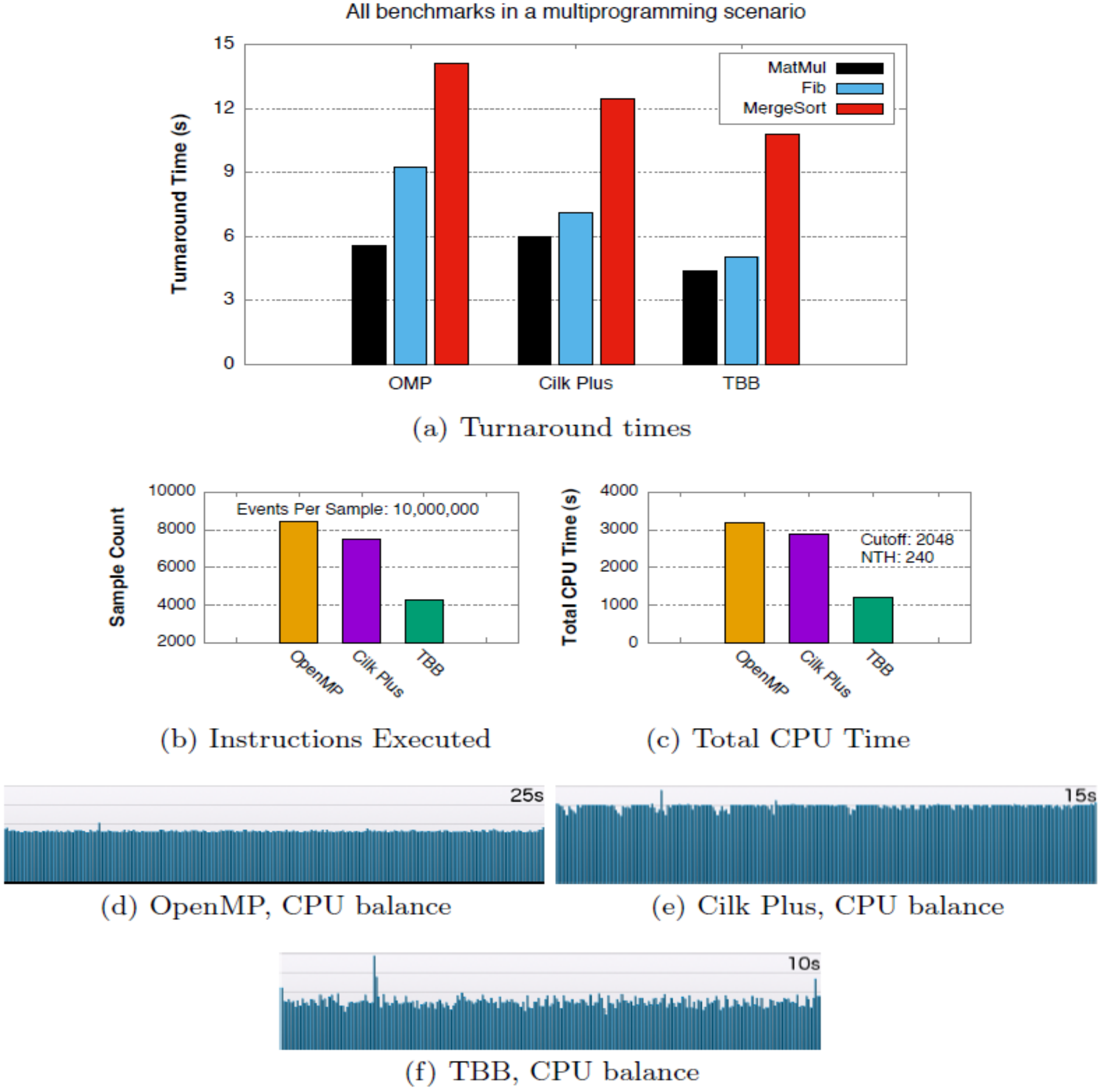
- **e.g. OpenMP**
  - Master thread is executing a serial region, slaves are spinning
  - A thread has finished a parallel region and is spinning in the barrier, waiting for others to reach that synchronisation point

Table 1. Percentage of the Total CPU Time consumed by the runtime libraries

Benchmark	**OpenMP** (`libiomp5.so`)	**Cilk Plus** (`libcilkrts.so.5`)	**TBB** (`libtbb.so.2`)
Fibonacci	50%	16%	5%
MergeSort	78%	81%	3%
MatMul	22% *(Dynamic)* 20% *(Static)*	6%	1%

How does it affect the performance when it comes to concurrent execution of the programs?

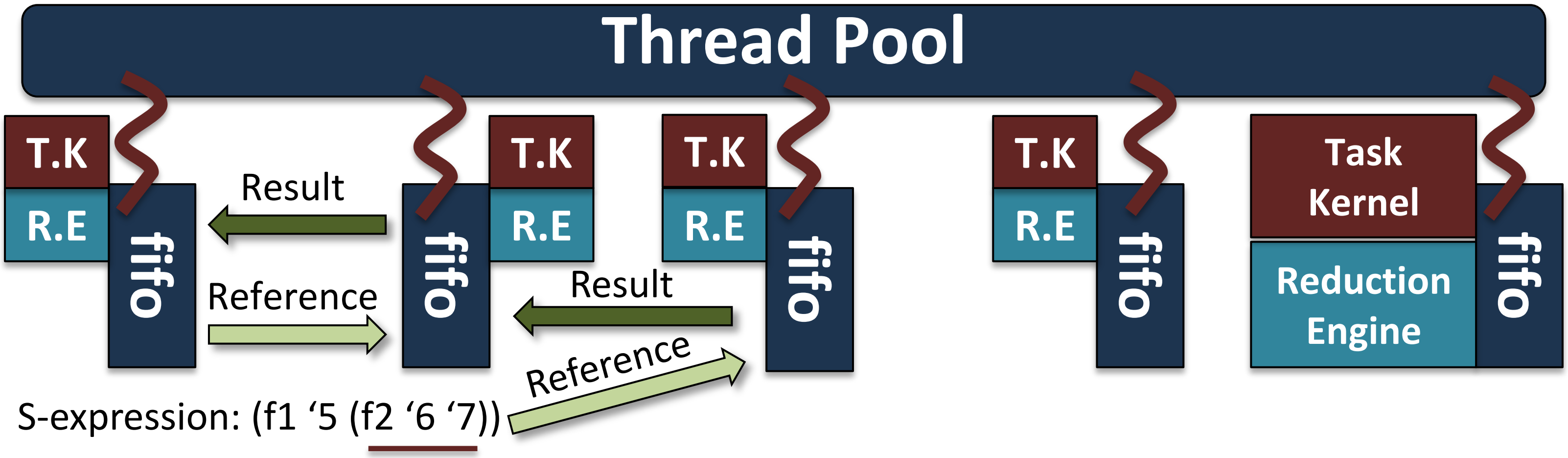
Multiprogramming



≈35% difference between the Total Turnaround Time of OpenMP and TBB

The Bigger Picture

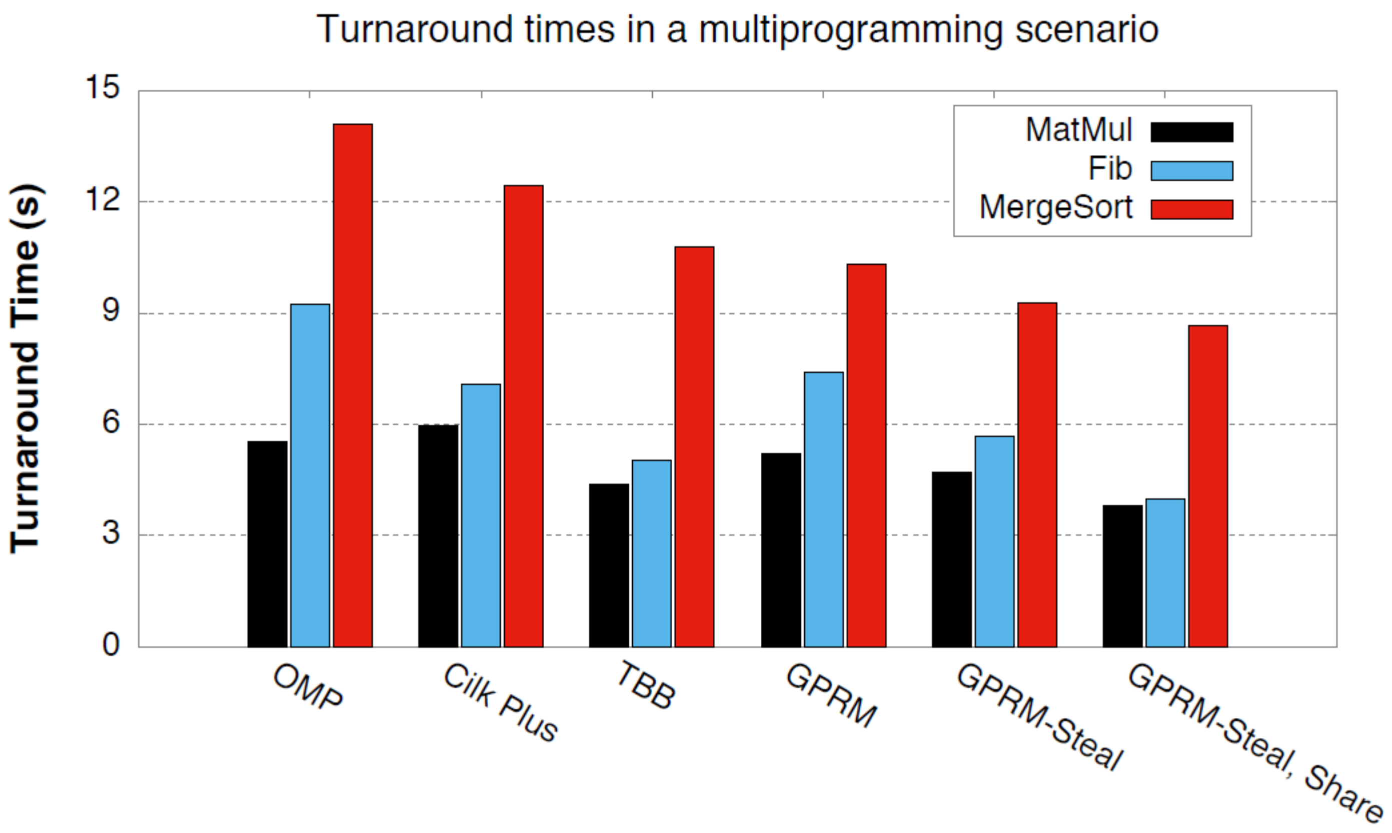
Glasgow Parallel Reduction Machine (GPRM)



- **Task Kernel:** A complex, self-contained unit which offers some specific functionalities to the system.
- **Task Manager (Reduction Engine):** Provides an interface to the *Task Kernel*. It is responsible for activating the corresponding task.

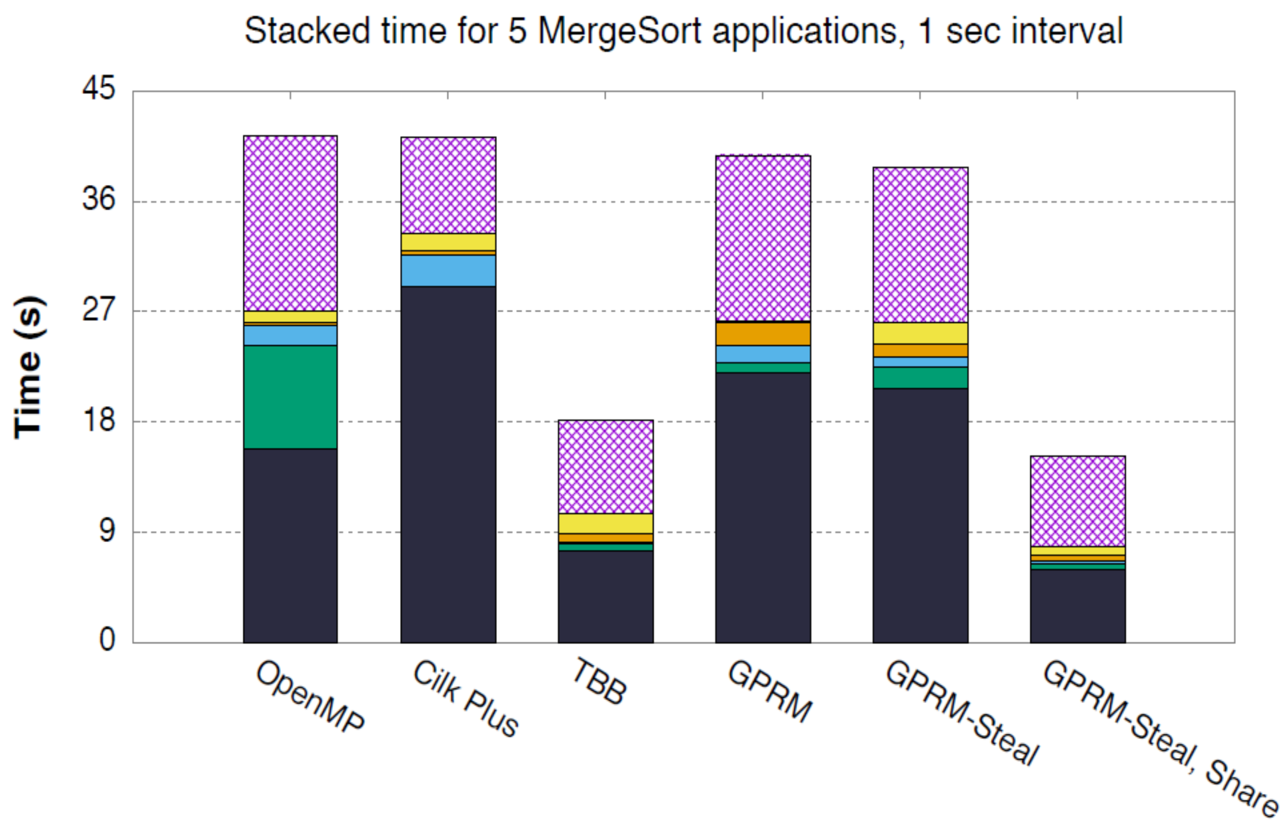
The combined operation of all Reduction Engines in all threads results in the parallel reduction of the entire program

The Bigger Picture
Glasgow Parallel Reduction Machine (GPRM)



The Bigger Picture

Glasgow Parallel Reduction Machine (GPRM)



University of Glasgow sicsa*

Conclusion

- **Comparison**
  - We have compared three popular parallel programming models on a modern manycore platform. Three benchmarks have exercised different aspects of the system performance
- **Performance Aspects**
  - Speedup by changing the number threads/tasks
  - **Total CPU Time**
  - CPU balance
  - Runtime systems overhead
- **Multiprogramming**
  - CPU time is precious!
  - Sharing some information between the programs present in the system can be useful
- **The Bigger Picture**
  - GPRM aims to provide an efficient parallel programming model for both single-programming and multiprogramming environments