Hybrid parallism on BG/Q with OpenMP

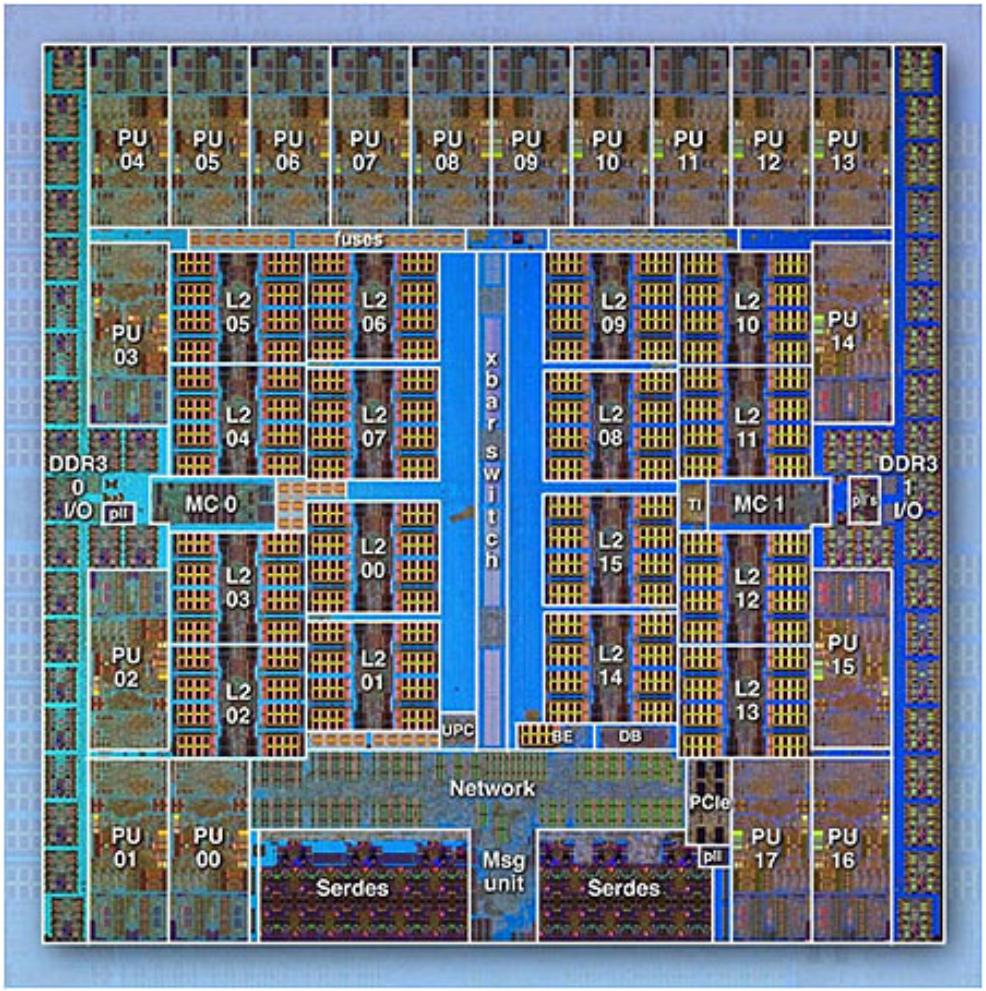
February 3, 2014 | T. Hater | EIC

Outline

- BG/Q hardware
- Shared memory parallelism
- OpenMP tutorial
- Hints for optimization
- BG/Q specific OpenMP topics

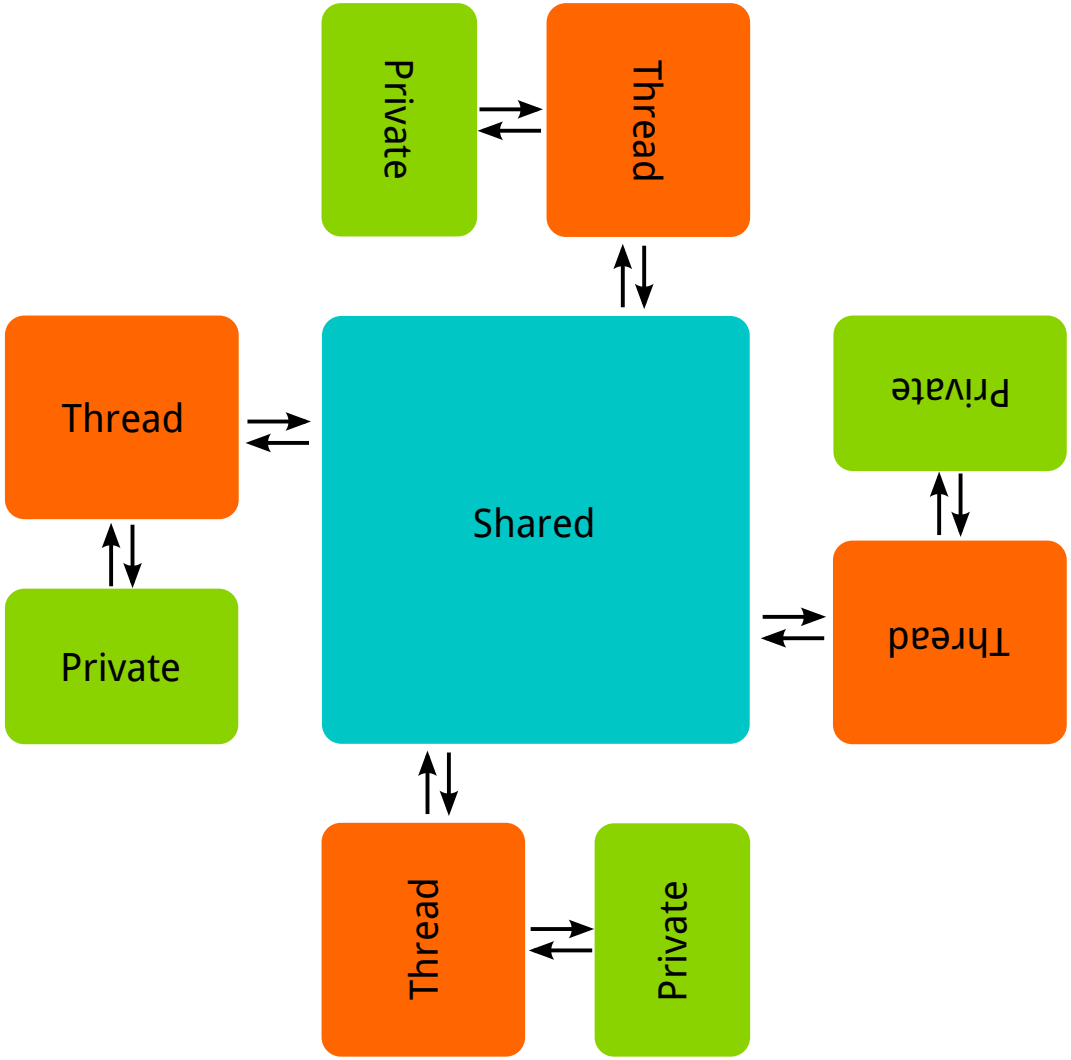
Motivation

The BG/Q Compute Chip



- SoC
- $1.6\,GHz$ in-order
- 16×4 threads
- 2 pipelines
- 200 $GFlop/s$
- 30 GB/s (42 GB/s)

Symmetric multiprocessing



OpenMP

- Regularly updated standard

OpenMP

- Regularly updated standard
- Supported by most major compilers

OpenMP

- Regularly updated standard
- Supported by most major compilers
- Platform independent

OpenMP

- Regularly updated standard
- Supported by most major compilers
- Platform independent
- Supports FORTRAN, C, C++

OpenMP

- Regularly updated standard
- Supported by most major compilers
- Platform independent
- Supports FORTRAN, C, C++
- Annotation based
  - Non-invasive
  - Incremental
  - Low development overhead

How does it work?

- Source code annotated by programmer

How does it work?

- Source code annotated by programmer
- Compiler generates calls to OMP runtime

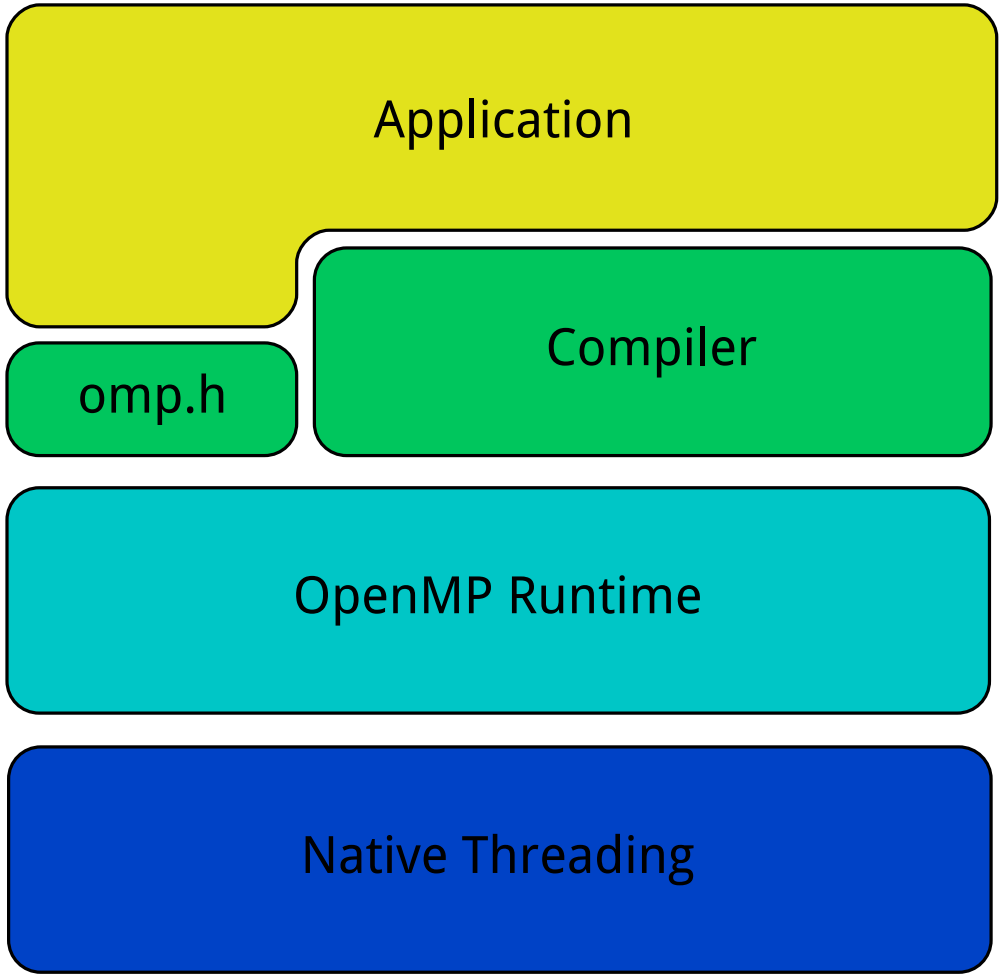
How does it work?

- Source code annotated by programmer
- Compiler generates calls to OMP runtime
- Runtime abstracts native threading

How does it work?

- Source code annotated by programmer
- Compiler generates calls to OMP runtime
- Runtime abstracts native threading
- Abstracts shared memory parallelism
  - False sharing
  - Race conditions

Organization



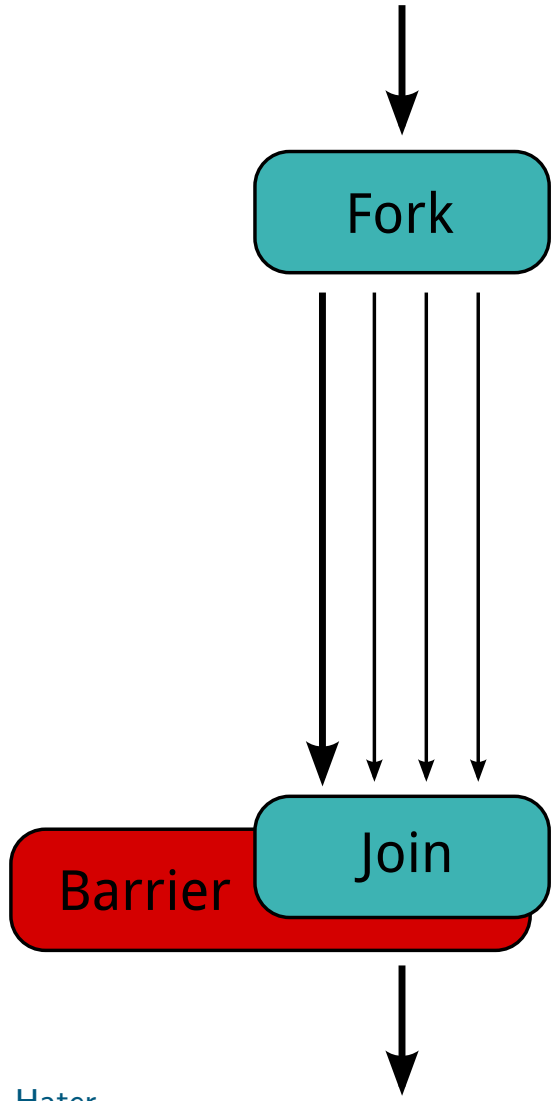
How do I parallelize my code?

1. Indentify bottleneck
2. Annotate hotspot
3. Profile for gains
4. Goto 1

OpenMP Introduction

The parallel region

```
#pragma omp parallel
{
   /* ... */
}
```

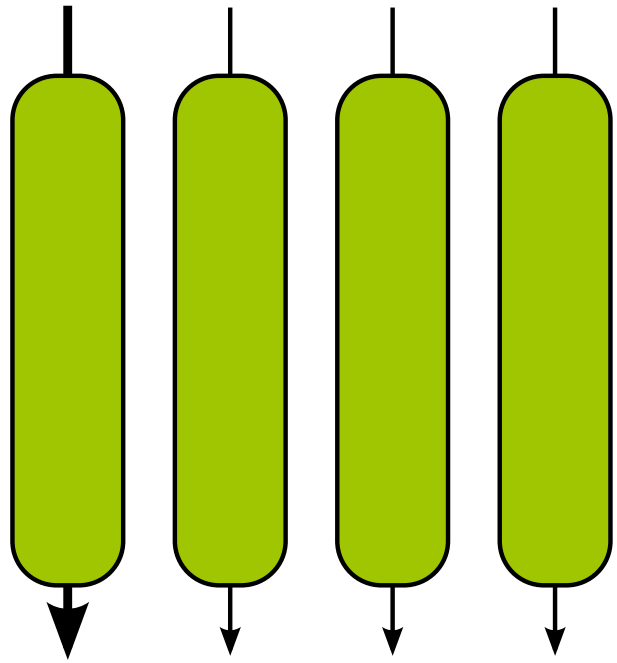


OpenMP Clauses

- Modify OpenMP statements
  `#pragma omp statement clause1(arg1,...)`
- Example: variable scope
  `shared/private(x,y,...)`
  `default(shared/private/...)`
- More available, depending on construct

The parallel loop

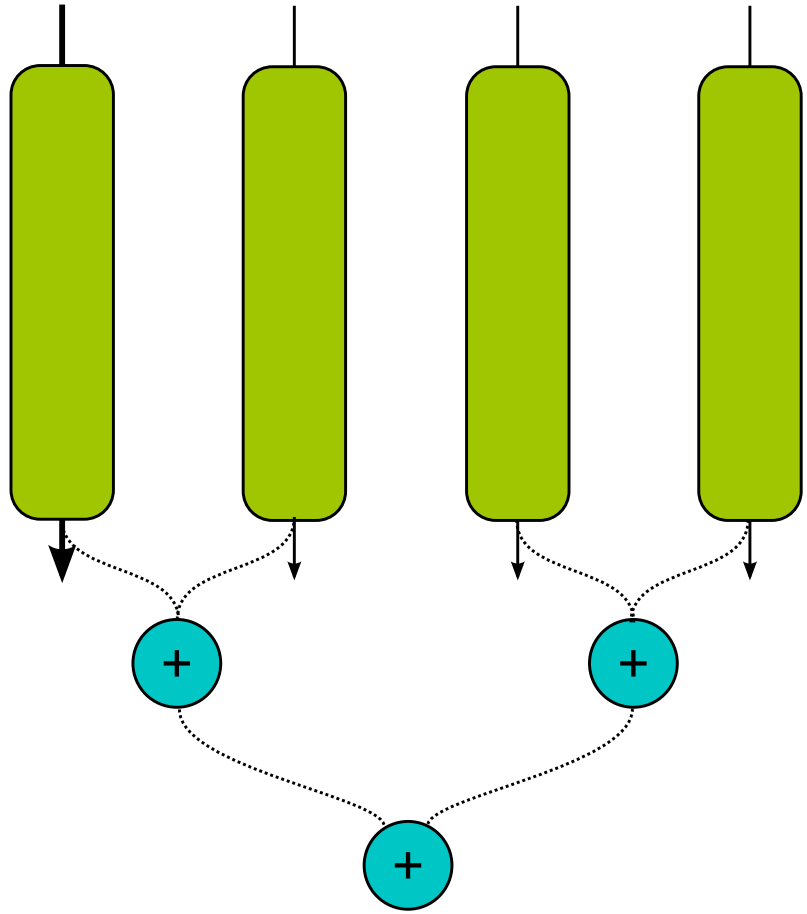
```
#pragma omp for
for(i = 1; i <= N; ++i)
{
   /* ... */
}
```



The parallel loop

Reduction

```
#pragma omp for reduction(+:c)
for(i = 0; i < N; ++i)
{
   c += a[i]
}
```



The parallel loop
Scheduling

- Parallel loops accept a `schedule(...,chunk)` clause

The parallel loop
Scheduling

- Parallel loops accept a `schedule(...,chunk)` clause
- Default to `static`: Divide iterations evenly

The parallel loop

Scheduling

- Parallel loops accept a `schedule(...,chunk)` clause
- Default to `static`: Divide iterations evenly
- If time per work item varies try `dynamic`.
  Each idle thread picks up a new chunk.

The parallel loop

Scheduling

- Parallel loops accept a `schedule(...,chunk)` clause
- Default to `static`: Divide iterations evenly
- If time per work item varies try `dynamic`.
  Each idle thread picks up a new chunk.
- If threads start at different times: `guided`.
  Like `dynamic`, but chunk size decreases exponentially.

Limiting parallelism

```
#pragma omp parallel
{
  #pragma omp single
  {
    // Arbitrary, but unique thread
  } // Barrier

  #pragma omp master
  {
    // Master thread
  } // No barrier
}
```

Ordering threads

```
#pragma omp parallel
{
  #pragma omp critical
  {
    // One thread at a time, arbitrary order
  } // Barrier

  #pragma omp atomic read|write|update|capture
    // Atomic access to memory
}
```

Sections

Static worksharing

```
#pragma omp sections
{
  #pragma omp section
  {
     // Thread 1
  }
  #pragma omp section
  {
     // Thread 2
  }
}
```

Explicit tasks

- Tasks are flexible building blocks for thread creation

Explicit tasks

- Tasks are flexible building blocks for thread creation
- `#pragma omp task` per encountering thread, create a task.

Explicit tasks

- Tasks are flexible building blocks for thread creation
- `#pragma omp task` per encountering thread, create a task.
  - Execution may commence in the creating thread.

Explicit tasks

- Tasks are flexible building blocks for thread creation
- `#pragma omp task` per encountering thread, create a task.
  - Execution may commence in the creating thread.

or

Explicit tasks

- Tasks are flexible building blocks for thread creation
- `#pragma omp task` per encountering thread, create a task.
  - Execution may commence in the creating thread.

  or
  - Deferred and picked by another thread in the team.

Explicit tasks

- Tasks are flexible building blocks for thread creation
- `#pragma omp task` per encountering thread, create a task.
  - Execution may commence in the creating thread.

  or
  - Deferred and picked by another thread in the team.
- `#pragma omp taskwait` waits for all sibling tasks.

Optimizing OpenMP

Fork/Join

- Fork/Join take time.
- May combine multiple regions.

```
#pragma omp parallel for
for(i=0; i<n; ++i) a[i] = 2*b[i];
c = 0;
#pragma omp parallel for reduction(+: c)
for(i=0; i<n; ++i) c += a[i];
```

Fork/Join

- Fork/Join take time.
- Combine multiple regions into one to amortize

```
#pragma omp parallel
{
  #pragma omp for
  for(i=0; i<n; ++i) a[i] = 2*b[i];
  c = 0;
  #pragma omp for reduction(+: c)
  for(i=0; i<n; ++i) c += a[i];
}
```

Barriers

- Many constructs feature implicit barriers

Barriers

- Many constructs feature implicit barriers
- If unneeded, they can be disabled using the `nowait` clause

```
#pragma omp parallel for nowait
for(i=0; i<n; ++i) {
  a[i] = 2*b[i];
}
```

Barriers

- Many constructs feature implicit barriers
- If unneeded, they can be disabled using the `nowait` clause

```
#pragma omp parallel for nowait
for(i=0; i<n; ++i) {
  a[i] = 2*b[i];
}
```

- Explicit barriers should be regarded with suspicion

Barriers

- Many constructs feature implicit barriers
- If unneeded, they can be disabled using the `nowait` clause

```
#pragma omp parallel for nowait
for(i=0; i<n; ++i) {
  a[i] = 2*b[i];
}
```

- Explicit barriers should be regarded with suspicion
- The same holds for `critical`

Barriers

- Many constructs feature implicit barriers
- If unneeded, they can be disabled using the `nowait` clause

```
#pragma omp parallel for nowait
for(i=0; i<n; ++i) {
  a[i] = 2*b[i];
}
```

- Explicit barriers should be regarded with suspicion
- The same holds for `critical`
- **But:** correctness comes first. Be careful.

Workloads

- OpenMP constructs are not for free

Workloads

- OpenMP constructs are not for free
- Amortize overhead by offering enough work

Workloads

- OpenMP constructs are not for free
- Amortize overhead by offering enough work
- Either number of iterations or effort per iteration

Workloads

- OpenMP constructs are not for free
- Amortize overhead by offering enough work
- Either number of iterations or effort per iteration
- If workload varies: try to tune `schedule`

OpenMP on BG/Q

Compiling for OpenMP

- Use the XL compilers
  `mpixlc_r`
  `mpixlcxx_r`
  `mpixlf_r`
  `mpixlf90_r`

Compiling for OpenMP

- Use the XL compilers
  `mpixlc_r`
  `mpixlcxx_r`
  `mpixlf_r`
  `mpixlf90_r`
- Add `-qsmp=omp` to compiler and linker flags.

Compiling for OpenMP

- Use the XL compilers
  `mpixlc_r`
  `mpixlcxx_r`
  `mpixlf_r`
  `mpixlf90_r`
- Add `-qsmp=omp` to compiler and linker flags.
- Automatically enables `-O2 -qhot`, suppress with `-qsmp=omp:noopt`.

Compiling for OpenMP

- Use the XL compilers
  `mpixlc_r`
  `mpixlcxx_r`
  `mpixlf_r`
  `mpixlf90_r`
- Add `-qsmp=omp` to compiler and linker flags.
- Automatically enables `-O2 -qhot`, suppress with `-qsmp=omp:noopt`.
- Automatically parallelizes on top of OpenMP if given `-qsmp`

XL OpenMP

- Standard version 3.1

XL OpenMP

- Standard version 3.1
- `OMP_NUM_THREADS` = `OMP_THREAD_LIMIT` = $\left\lfloor \frac{64}{RanksPerNode} \right\rfloor$

XL OpenMP

- Standard version 3.1
- `OMP_NUM_THREADS` = `OMP_THREAD_LIMIT` = $\left\lfloor \frac{64}{RanksPerNode} \right\rfloor$
- May oversubscribe, but be careful.

XL OpenMP

- Standard version 3.1
- `OMP_NUM_THREADS` = `OMP_THREAD_LIMIT` = $\left\lfloor \frac{64}{RanksPerNode} \right\rfloor$
- May oversubscribe, but be careful.
- `OMP_PROC_BIND` = `True` due to CNK limitation

XL OpenMP

- Standard version 3.1
- `OMP_NUM_THREADS` = `OMP_THREAD_LIMIT` = $\left\lfloor \frac{64}{RanksPerNode} \right\rfloor$
- May oversubscribe, but be careful.
- `OMP_PROC_BIND` = `True` due to CNK limitation
- No nested OpenMP

Exploiting BG/Q features

- `omp barrier` and `lock` use L2 atomics.

Exploiting BG/Q features

- `omp barrier` and `lock` use L2 atomics.
- `omp atomic` exploits hardware atomics

Exploiting BG/Q features

- `omp barrier` and `lock` use L2 atomics.
- `omp atomic` exploits hardware atomics
- Waiting threads go to sleep.

Exploiting BG/Q features

- `omp barrier` and `lock` use L2 atomics.
- `omp atomic` exploits hardware atomics
- Waiting threads go to sleep.
- Thread local storage using
  `#pragma ibm threadlocal`

Exploiting BG/Q features

- `omp barrier` and `lock` use L2 atomics.
- `omp atomic` exploits hardware atomics
- Waiting threads go to sleep.
- Thread local storage using
  `#pragma ibm threadlocal`
- For hard to parallelize loops: speculation & TM (later talk)

Overheads

Experimental values

- $\{\texttt{lock|barrier|critical}\} \leq 1\mu s$

Overheads

Experimental values

- $\{\texttt{lock|barrier|critical}\} \leq 1\mu s$
- `parallel` $1\mu s$ (1 thread) – $50\mu s$ (64 threads)

Overheads

Experimental values

- $\{\texttt{lock|barrier|critical}\} \leq 1\mu s$
- `parallel` $1\mu s$ (1 thread) – $50\mu s$ (64 threads)
- `for` loops $1\mu s$ (1 thread) – $50\mu s$ (64 threads)

Overheads

Experimental values

- $\{$`lock|barrier|critical`$\} \leq 1\mu s$
- `parallel` $1\mu s$ (1 thread) – $50\mu s$ (64 threads)
- `for` loops $1\mu s$ (1 thread) – $50\mu s$ (64 threads)
- Task $\{$`create|wait`$\}$ $2\mu s$ (1 thread) – $50\mu s$ (16 threads)

Overheads

Experimental values

- $\{$`lock|barrier|critical`$\} \leq 1\mu s$
- `parallel` $1\mu s$ (1 thread) – $50\mu s$ (64 threads)
- `for` loops $1\mu s$ (1 thread) – $50\mu s$ (64 threads)
- Task $\{$`create|wait`$\}$ $2\mu s$ (1 thread) – $50\mu s$ (16 threads)

Rule of thumb

$100\mu s$ for tasks, $10\mu s$ for loops and $1\mu s$ everything else.

Tuning

- Experiment with $n_{Threads} : n_{RanksPerNode}$

Tuning

- Experiment with $n_{Threads} : n_{RanksPerNode}$
- `BG_SMP_FAST_WAKEUP=YES` + `OMP_WAIT_POLICY=active`

Tuning

- Experiment with $n_{Threads} : n_{RanksPerNode}$
- `BG_SMP_FAST_WAKEUP=YES` + `OMP_WAIT_POLICY=active`
- Keep loops small if ≥ 16 Threads (L1 cache)

Tuning

- Experiment with $n_{Threads} : n_{RanksPerNode}$
- `BG_SMP_FAST_WAKEUP=YES` + `OMP_WAIT_POLICY=active`
- Keep loops small if ≥ 16 Threads (L1 cache)
- Normally `static` scheduling is sensible

Tuning

- Experiment with $n_{Threads} : n_{RanksPerNode}$
- `BG_SMP_FAST_WAKEUP=YES` + `OMP_WAIT_POLICY=active`
- Keep loops small if ≥ 16 Threads (L1 cache)
- Normally `static` scheduling is sensible
- On *simple* loop nests, try `collapse(n)`

Tuning

- Experiment with $n_{Threads} : n_{RanksPerNode}$
- `BG_SMP_FAST_WAKEUP=YES` + `OMP_WAIT_POLICY=active`
- Keep loops small if ≥ 16 Threads (L1 cache)
- Normally `static` scheduling is sensible
- On *simple* loop nests, try `collapse(n)`
- Do not use `strict` math for reductions

MPI Comm threads

- BG/Q can use idle threads for asynchronous progress on MPI

MPI Comm threads

- BG/Q can use idle threads for asynchronous progress on MPI
- Initialize `MPI_Init_thread(..., MPI_THREAD_MULTIPLE)` (`PAMID_CONTEXT_POST=1 PAMID_ASYNC_PROGRESS=1`)

MPI Comm threads

- BG/Q can use idle threads for asynchronous progress on MPI
- Initialize `MPI_Init_thread(..., MPI_THREAD_MULTIPLE)` (`PAMID_CONTEXT_POST=1 PAMID_ASYNC_PROGRESS=1`)
- Compile with `-qsmp`

MPI Comm threads

- BG/Q can use idle threads for asynchronous progress on MPI
- Initialize `MPI_Init_thread(..., MPI_THREAD_MULTIPLE)` (`PAMID_CONTEXT_POST=1 PAMID_ASYNC_PROGRESS=1`)
- Compile with `-qsmp`
- The rest *should* happen automagically

Closing words

- Extremely fast and incomplete introduction to OpenMP.
- Deceptively easy looking!
- We did not talk about memory consistency.
- On JUQUEEN you *must* go hybrid.

Resources

- XL compiler manuals
  `http://pic.dhe.ibm.com/infocenter/compbg/v121v141/`
- OpenMP standard
  `http://www.openmp.org/mp-documents/OpenMP3.1.pdf`
- OpenMP overview card
  `http://openmp.org/mp-documents/`
  `OpenMP3.1-CCard.pdf`
  `OpenMP3.1-FortranCard.pdf`