A Data-flow Approach to Solving the von Neumann Bottlenecks

Antoniu Pop

Inria and École normale supérieure, Paris

St. Goar, June 21, 2013
Problem Statement
“Single-threaded” performance still improving...

J. Preshing. A Look Back at Single-Threaded CPU Performance. preshing.com
Von Neumann Bottlenecks — Sequential Context

**Program counter**
Control-flow drives execution

**Shared memory**

**Impact**
All latency is paid in full

**Hardware**
Out-of-order execution

**Compiler**
VLIW

**Software**
Coroutines

Data flow principles: local/private memory, data availability drives execution
### Von Neumann Bottlenecks — Parallel Context

<table>
<thead>
<tr>
<th>Program counter</th>
<th>Shared memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control-flow drives execution</td>
<td>Memory consistency constraints</td>
</tr>
</tbody>
</table>

#### Impact
- All latency is paid in full
- **Higher latency:**
  - memory subsystem overload
  - bandwidth
  - NUMA
  - program dependences

#### Hardware
- Out-of-order execution
- **Relaxed memory models**
  - Scratchpads, non-coherent caches

#### Compiler
- VLIW
- **Intrinsics, low-level atomics in C/C++11**

#### Software
- Coroutines
- **Data flow programming models**

#### Programmability setback: programmers are directly exposed to hardware complexity
Open source implem. in

GCC 4.7

~30 kLoC

GCC summit '08, '09

http://openstream.info

OpenStream

ACM TACO'13, IJPP'11, HiPEAC'11
Foster collaborations and technology transfers
IBM Haifa, U. of Sienna, Thales CS, Thales RT, CAPS Enterprise, Kalray, UPMC, BSC

Open source implem. in
GCC 4.7
~30 kLoC
GCC summit '08, '09

OpenStream
http://openstream.info
ACM TACO'13, IJPP'11, HiPEAC'11
Foster collaborations and technology transfers
IBM Haifa, U. of Sienna, Thales CS, Thales RT, CAPS Enterprise, Kalray, UPMC, BSC

Open source implem. in
GCC4.7
~30 kLoC
GCC summit '08, '09

New stream synchronization algorithms
CASES'10, submitted EMSOF'T13

Benchmarks ~27 kLoC
with third party contributions

Profiling and visualiz. infrastr.
collaboration with UPMC

OpenStream
http://openstream.info

ACM TACO'13, IJPP'11, HiPEAC'11
OpenStream

OpenMP extension

- leverage existing toolchains and knowledge
- maximize productivity
OpenStream

OpenMP extension

- leverage existing toolchains and knowledge
- maximize productivity

Parallelize irregular, dynamic codes

- no static/periodic behavior required
- maximize expressiveness
OpenStream

OpenMP extension
- leverage existing toolchains and knowledge
- maximize productivity

Parallelize irregular, dynamic codes
- no static/periodic behavior required
- maximize expressiveness

Mitigate the von Neumann bottlenecks
- decoupled producer/consumer pipelines
- maximize efficiency
OpenStream Introductory Example

\begin{verbatim}
for (i = 0; i < N; ++i) {

    #pragma omp task firstprivate (i) output (x) // T1
    x = foo (i);

    #pragma omp task input (x)                   // T2
    print (x);
}
\end{verbatim}
OpenStream Introductory Example

```
for (i = 0; i < N; ++i) {
    #pragma omp task firstprivate (i) output (x) // T1
    x = foo (i);

    #pragma omp task input (x)            // T2
    print (x);
}
```

**Control program** sequentially creates $N$ instances of $T1$ and of $T2$

**Firstprivate clause** privatizes variable $i$ with initialization at task creation

**Output clause** gives write access to stream $x$

**Input clause** gives read access to stream $x$

**Stream $x$** has FIFO semantics
Stream FIFO Semantics

```c
#pragma omp task output (x) // Task T1
    x = ...;
for (i = 0; i < N; ++i) {
    int window_a[2], window_b[2];

    #pragma omp task output (x << window_a[2]) // Task T2
        window_a[0] = ...; window_a[1] = ...;
    if (i % 2) {
        #pragma omp task input (x >> window_b[2]) // Task T3
            use (window_b[0], window_b[1]);
    }
    #pragma omp task input (x) // Task T4
        use (x);
}
```
Stream FIFO Semantics

```c
#pragma omp task output (x) // Task T1
    x = ...;
for (i = 0; i < N; ++i) {
    int window_a[2], window_b[2];

    #pragma omp task output (x << window_a[2]) // Task T2
        window_a[0] = ...; window_a[1] = ...;
    if (i % 2) {
        #pragma omp task input (x >> window_b[2]) // Task T3
            use (window_b[0], window_b[1]);
    }
    #pragma omp task input (x) // Task T4
        use (x);
}
```
Stream FIFO Semantics

```c
#pragma omp task output (x) // Task T1
    x = ...;
for (i = 0; i < N; ++i) {
    int window_a[2], window_b[2];

    #pragma omp task output (x << window_a[2]) // Task T2
        window_a[0] = ...; window_a[1] = ...;
    if (i % 2) {
        #pragma omp task input (x >> window_b[2]) // Task T3
            use (window_b[0], window_b[1]);
    }
    #pragma omp task input (x) // Task T4
        use (x);
}
```

Interleaving of stream accesses
Foster collaborations and technology transfers
IBM Haifa, U. of Sienna, Thales CS, Thales RT, CAPS Enterprise, Kalray, UPMC, BSC

<table>
<thead>
<tr>
<th>Open source implem. in GCC4.7</th>
<th>New stream synchronization algorithms</th>
<th>Control-Driven Data Flow CDDF</th>
<th>Benchmarks ~27 kLoC</th>
<th>Profiling and visualiz. infrastr. collaboration with UPMC</th>
</tr>
</thead>
<tbody>
<tr>
<td>GCC summit '08, '09</td>
<td>CASES'10, submitted EMSOFT'13</td>
<td>PhD thesis, submitted research report</td>
<td>with third party contributions</td>
<td>with third party contributions</td>
</tr>
</tbody>
</table>

OpenStream
http://openstream.info

ACM TACO'13, IJPP'11, HiPEAC'11
Control-Driven Data Flow

Define the formal semantics of imperative programming languages with dynamic, dependent task creation

- control flow: dynamic construction of task graphs
- data flow: decoupling dependent computations (Kahn)
CDDF model

• Control program
  • imperative program
  • creates tasks
  • model: execution graph of activation points, each generating a task activation

• Tasks
  • imperative program with a dynamic stream access signature
  • becomes executable once its dependences are satisfied
  • recursively becomes the control program for tasks created within
    • work in progress
    • link with synchronous languages
  • model: task activation defined as a set of stream accesses

• Streams
  • Kahn-like unbounded, indexed channels
  • multiple producers and/or consumers
  • specify dependences and/or communication
  • model: indexed set of memory locations, defined on a finite subset
Control-Driven Data Flow – Results

• Deadlock classification
  • insufficiency deadlock: missing producer before a barrier or control program termination
  • functional deadlock: dependence cycle
  • spurious deadlock: deadlock induced by CDDF semantics on dependence enforcement (Kahn prefixes)

• Conditions on program state allowing to prove
  • deadlock freedom
  • compile-time serializability
  • functional and deadlock determinism

<table>
<thead>
<tr>
<th>Condition on state ( \sigma = (k_e, A_e, A_o) )</th>
<th>Deadlock Freedom properties</th>
<th>Serializability</th>
<th>Determinism Func\textsuperscript{al} &amp; Deadlock</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \neg D(\sigma) ) \wedge \forall s, \neg MPMC(s) ) \ Weaker than Kahn monotonicity</td>
<td>no</td>
<td>no</td>
<td>yes if ( \neg ID(\sigma) )</td>
</tr>
<tr>
<td>( SCC(H(\sigma)) = \emptyset ) \ Common case, static over-approx.</td>
<td>no</td>
<td>no</td>
<td>yes if ( \neg ID(\sigma) )</td>
</tr>
<tr>
<td>( SC(\sigma) ) \wedge \Omega(k_e) \in \Pi ) \ Less restrictive than strictness</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>( \forall \sigma, SC(\sigma) ) \ Relaxed strictness</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>

SCC(H(\sigma)) = \emptyset

SCC(H(\sigma)) = \emptyset

Weaker than Kahn monotonicity

Common case, static over-approx.

Less restrictive than strictness

Relaxed strictness
**Foster collaborations and technology transfers**

IBM Haifa, U. of Sienna, Thales CS, Thales RT, CAPS Enterprise, Kalray, UPMC, BSC

<table>
<thead>
<tr>
<th>Open source implement. in GCC4.7 ~30 kLoC</th>
<th>New stream synchronization algorithms</th>
<th>Feed-Forward Data Flow</th>
<th>Control-Driven Data Flow CDDF</th>
</tr>
</thead>
<tbody>
<tr>
<td>GCC summit '08, '09</td>
<td>CASES'10, submitted EMSOFT'13</td>
<td></td>
<td>PhD thesis, submitted research report</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Benchmarks ~27 kLoC with third party contributions</th>
<th>Profiling and visualiz. infrastr. collaboration with UPMC</th>
</tr>
</thead>
</table>

**OpenStream**

http://openstream.info

ACM TACO'13, IJPP'11, HiPEAC'11
Runtime Design and Implementation

Lessons learned...
Lessons learned...

1. eliminate false sharing
Runtime Design and Implementation

Lessons learned...

1. eliminate false sharing
2. use software caching to reduce cache traffic
Runtime Design and Implementation

Lessons learned...

1. eliminate false sharing
2. use software caching to reduce cache traffic
3. avoid atomic operations on data that is effectively shared across many cores
Runtime Design and Implementation

Lessons learned...

1. eliminate false sharing
2. use software caching to reduce cache traffic
3. avoid atomic operations on data that is effectively shared across many cores
4. avoid effective sharing of concurrent structures
Feed-Forward Data Flow

1. Resolve dependences at task creation
2. Link forward: producers know their consumers before executing
Feed-Forward Data Flow

1. Resolve dependences at task creation
2. Link forward: producers know their consumers before executing
Feed-Forward Data Flow

1. Resolve dependences at task creation
2. Link forward: producers know their consumers before executing
3. Local work-stealing queue for ready tasks
Feed-Forward Data Flow

1. Resolve dependences at task creation
2. Link forward: producers know their consumers before executing
3. Local work-stealing queue for ready tasks
4. Producer decides which consumers become executable
   - local consensus among producers providing data to the same task
   - without traversing effectively shared data structures
Comparison to StarSs: Block-sparse LU factorization
### Foster collaborations and technology transfers

IBM Haifa, U. of Sienna, Thales CS, Thales RT, CAPS Enterprise, Kalray, UPMC, BSC

<table>
<thead>
<tr>
<th>Open source implem. in GCC4.7 ~30 kLoC</th>
<th>New stream synchronization algorithms</th>
<th>Feed-Forward Data Flow</th>
<th>Control-Driven Data Flow CDDF</th>
<th>Proof techniques for concurrent algorithms</th>
<th>Benchmarks ~27 kLoC with third party contributions</th>
<th>Profiling and visualiz. infrastr.</th>
</tr>
</thead>
<tbody>
<tr>
<td>GCC summit '08, '09</td>
<td>CASES'10, submitted EMSOFT'13</td>
<td></td>
<td>PhD thesis, submitted research report</td>
<td>PPoPP'13</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**OpenStream**

*http://openstream.info*

*ACM TACO'13, IJPP'11, HiPEAC'11*
Correct, Efficient yet Relaxed

Hardware mitigation of the shared memory bottleneck: relax memory consistency

- impacts many programmers
- difficult to reason about program execution
Correct, Efficient yet Relaxed

Hardware mitigation of the shared memory bottleneck: relax memory consistency
• impacts many programmers
• difficult to reason about program execution

Contributions
• first application of a formal relaxed memory model to the manual proof of real-world, performance-critical algorithms
• efficient implementations in C11 and inline assembly of Chase&Lev work-stealing and an optimized lock-free FIFO queue
• proof blueprints
Memory Consistency

<table>
<thead>
<tr>
<th>core 0</th>
<th>core 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>[x] = 1</td>
<td>[y] = 1</td>
</tr>
<tr>
<td>r0 = [y]</td>
<td>r1 = [x]</td>
</tr>
</tbody>
</table>

main memory

[x] = 0
[y] = 0
Memory Consistency

- Sequential consistency: behavior equivalent to serial interleaving of accesses.
  - Will necessarily read $r0 = 1$ or $r1 = 1$
Memory Consistency

- Sequential consistency: behavior equivalent to serial interleaving of accesses.
- Total Store Order (x86): write buffer delays visibility of stores from other processors
Memory Consistency

- Sequential consistency: behavior equivalent to serial interleaving of accesses.

- Total Store Order (x86): write buffer delays visibility of stores from other processors

Additional TSO "perceived" interleavings... all permutations
Memory Consistency

- Sequential consistency: behavior equivalent to serial interleaving of accesses.

- Total Store Order (x86): write buffer delays visibility of stores from other processors

Sequential consistency possible interleavings

```
x = 1
y = 1
r0 = [y]
r1 = [x]
```

Additional TSO "perceived" interleavings... all permutations

```
r0 = [y]
r1 = [x]
x = 1
y = 1
```

```
r0 = 0
r1 = 1
x = 0
y = 0
```

```
r0 = 1
r1 = 1
x = 1
y = 1
```

```
r0 = 1
r1 = 0
x = 1
y = 0
```

```
r0 = 1
r1 = 1
x = 1
y = 1
```

```
r0 = 0
r1 = 0
x = 0
y = 0
```

```
r0 = 1
r1 = 0
x = 1
y = 0
```

```
r0 = 0
r1 = 1
x = 0
y = 1
```

```
r0 = 1
r1 = 1
x = 1
y = 1
```

```
r0 = 0
r1 = 0
x = 0
y = 0
```
Memory Consistency

- Sequential consistency: behavior equivalent to serial interleaving of accesses.

- Total Store Order (x86): write buffer delays visibility of stores from other processors
  - Can read $r0 = 0$ and $r1 = 0$
Memory Consistency

- Sequential consistency: behavior equivalent to serial interleaving of accesses.

- Total Store Order (x86): write buffer delays visibility of stores from other processors
  - Can read $r_0 = 0$ and $r_1 = 0$
Memory Consistency

- Sequential consistency: behavior equivalent to serial interleaving of accesses.
  - Will necessarily read $r_0 = 1$ or $r_1 = 1$

- Total Store Order (x86): write buffer delays visibility of stores from other processors
  - Can read $r_0 = 0$ and $r_1 = 0$

Sequential consistency possible interleavings

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_0 = [y]$</td>
<td>$r_0 = [y]$</td>
<td>$r_0 = [y]$</td>
<td>$r_0 = [y]$</td>
<td>$r_0 = [y]$</td>
<td>$r_0 = [y]$</td>
</tr>
<tr>
<td>$r_1 = [x]$</td>
<td>$r_1 = [x]$</td>
<td>$r_1 = [x]$</td>
<td>$r_1 = [x]$</td>
<td>$r_1 = [x]$</td>
<td>$r_1 = [x]$</td>
</tr>
</tbody>
</table>

Additional TSO "perceived" interleavings... all permutations

<table>
<thead>
<tr>
<th>$r_0 = [y]$</th>
<th>$r_0 = [y]$</th>
<th>$r_0 = [y]$</th>
<th>$r_0 = [y]$</th>
<th>$r_0 = [y]$</th>
<th>$r_0 = [y]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_1 = [x]$</td>
<td>$r_1 = [x]$</td>
<td>$r_1 = [x]$</td>
<td>$r_1 = [x]$</td>
<td>$r_1 = [x]$</td>
<td>$r_1 = [x]$</td>
</tr>
</tbody>
</table>

[x] = 1
[r0 = [y]]
r1 = [x]
[y] = 1
[x] = 1
[y] = 1

[43]
Memory Consistency

- **Sequential consistency**: behavior equivalent to serial interleaving of accesses.

- **Total Store Order (x86)**: write buffer delays visibility of stores from other processors
  - Can read \( r0 = 0 \) and \( r1 = 0 \)
Memory Consistency

- Sequential consistency: behavior equivalent to serial interleaving of accesses.

- Total Store Order (x86): write buffer delays visibility of stores from other processors
  - Can read $r0 = 0$ and $r1 = 0$

Sequential consistency possible interleavings

```
core 0
[x] = 1
r0 = [y]

[y] = 1
r1 = [x]

[x] = 0
[y] = 0
```

```
r0 = 0           r0 = [y]        r0 = [y]        r0 = [y]        r0 = [y]        r0 = [y]        r0 = [y]
r1 = [x]         r1 = [x]        r1 = [x]        r1 = [x]        r1 = [x]        r1 = [x]        r1 = [x]
```

Additional TSO "perceived" interleavings... all permutations

```
Sequence          r0 = [y]
[x] = 1
r1 = [x]
[y] = 1
r0 = 0
r1 = 0
```

```
[x] = 1
[y] = 1
r0 = [y]
[ ] = 1
r1 = [x]
```

- goes to write buffer
- goes to write buffer
- reads 0 from memory
- reads 0 from memory
Memory Consistency

- Sequential consistency: behavior equivalent to serial interleaving of accesses.

- Total Store Order (x86): write buffer delays visibility of stores from other processors
  - Can read $r0 = 0$ and $r1 = 0$

### Sequential consistency possible interleavings

- $[x] = 1$
- $r0 = [y]$
- $r1 = [x]$

### Additional TSO "perceived" interleavings... all permutations

- $r0 = [y]$
- $r1 = [x]$
- $[x] = 1$
- $[y] = 1$
- $r0 = 0$
- $r1 = 0$

- Goes to write buffer
- Reads 0 from memory
- Write buffer of core 0 flushes to memory
Memory Consistency

- Sequential consistency: behavior equivalent to serial interleaving of accesses.

- Total Store Order (x86): write buffer delays visibility of stores from other processors
  - Can read \( r0 = 0 \) and \( r1 = 0 \)

\[
\begin{align*}
&\text{core 0} & \quad &\text{core 1} \\
& [x] = 1 & & [y] = 1 \\
& r0 = [y] & & r1 = [x] \\
\end{align*}
\]

Sequential consistency possible interleavings

- \( [x] = 1 \)
  - \( r0 = [y] \)
  - \( r1 = [x] \)

- \( [y] = 1 \)
  - \( r0 = [y] \)
  - \( r1 = [x] \)

- \( r0 = 0 \)
  - \( r1 = 1 \)

Additional TSO "perceived" interleavings... all permutations

- \( r0 = [y] \)
  - \( r1 = [x] \)
  - \( [x] = 1 \)
  - \( [y] = 1 \)
  - \( r0 = [y] \)
  - \( r1 = [y] \)
  - \( r0 = [y] \)
  - \( r1 = [x] \)

- Write buffer of core 0 flushes to memory
- Write buffer of core 1 flushes to memory
- Goes to write buffer
- Reads 0 from memory

- \( r0 = 0 \)
  - \( r1 = 0 \)
Memory Consistency and Relaxation

Sequential consistency – interleaving

+ total order of all memory operations

- no longer a valid hypothesis: performance bottleneck
Memory Consistency and Relaxation

Sequential consistency – interleaving
+ total order of all memory operations
  – no longer a valid hypothesis: performance bottleneck

Total store order (x86)
+ total order of stores: reason about global invariants
  – does not scale well
Memory Consistency and Relaxation

Sequential consistency – interleaving
- total order of all memory operations
- no longer a valid hypothesis: performance bottleneck

Total store order (x86)
- total order of stores: reason about global invariants
- does not scale well

POWER, ARM, C/C++11...
- partial order of memory operations
  - different processors may have conflicting views of memory
- better scalability at a lower power price tag
Work-stealing performance on Tegra 3, 4-core ARM
int steal(Deque *q) {
    size_t b = load_explicit(&q->bottom, relaxed) - 1;
    Array *a = load_explicit(&q->array, relaxed);
    store_explicit(&q->bottom, b, relaxed);
    thread_fence(seq_cst);
    size_t t = load_explicit(&q->top, relaxed);
    int x = EMPTY;
    if (t <= b) {
        /* Non-empty queue. */
        Array *a = load_explicit(&q->array, consume);
        x = load_explicit(&a->buffer[b % a->size], relaxed);
        if (t == b) {
            /* Single last element in queue. */
            if (!compare_exchange_strong_explicit
                (&q->top, &t, t + 1, seq_cst, relaxed))
                /* Failed race. */
                x = EMPTY;
                store_explicit(&q->bottom, b + 1, relaxed);
            }
        } else { /* Empty queue. */
            x = EMPTY;
            store_explicit(&q->bottom, b + 1, relaxed);
        }
        return x;
    }
}

void push(Deque *q, int x) {
    size_t b = load_explicit(&q->bottom, relaxed);
    size_t t = load_explicit(&q->top, acquire);
    Array *a = load_explicit(&q->array, relaxed);
    if (b - t > a->size - 1) { /* Full queue. */
        resize(q);
        a = load_explicit(&q->array, relaxed);
    }
    store_explicit(&a->buffer[b % a->size], x, relaxed);
    thread_fence(release);
    store_explicit(&q->bottom, b + 1, relaxed);
}
Proof method for relaxed memory consistency

Reason on partial order relations resulting from the memory ordering constraints enforced in a given memory model.

1. Formalize an undesirable property (e.g., task read twice, task lost...)
2. Find the corresponding memory events
3. Prove that they conflict with the partial order relations enforced by barriers and atomic instructions present in the code.
Foster collaborations and technology transfers
IBM Haifa, U. of Sienna, Thales CS, Thales RT, CAPS Enterprise, Kalray, UPMC, BSC

Open source implem. in
GCC4.7
~30 kLoC
GCC summit '08, '09

New stream synchronization algorithms
CASES'10, submitted EMSOFT'13

Feed-Forward Data Flow

Control-Driven Data Flow
CDDF
PhD thesis, submitted research report

Proof techniques for concurrent algorithms
PPoPP'13

Benchmarks
~27 kLoC
with third party contributions

Profiling and visualiz. infrastr.
collaboration with UPMC

OpenStream
http://openstream.info
ACM TACO'13, IJPP'11, HiPEAC'11
Foster collaborations and technology transfers
IBM Haifa, U. of Sienna, Thales CS, Thales RT, CAPS Enterprise, Kalray, UPMC, BSC

Open source implem. in
GCC4.7 ~30 kLoC
GCC summit '08, '09

New stream synchro-
ization algorithms
CASES'10, submitted EMSOF'T13

Feed-
Forward Data Flow

Control-
Driven
Data Flow
CDDF
PhD thesis, submitted research report

Proof techniques for concurrent algorithms
PPoPP'13

Bench-
marks ~27 kLoC
with third party contributions

Profiling and visualiz. infrastr.
collaboration with UPMC

http://openstream.info

OpenStream
ACM TACO'13, IJPP'11, HiPEAC'11

Automatic extraction of data flow threads from imperative programs (IEEE Micro'12)
Compiler-Aided Parallelization

Automatic extraction of data flow threads from imperative C programs

- direct automatic parallelization
- exploit additional parallelism at finer granularity within OpenStream tasks
Compiler-Aided Parallelization

Automatic extraction of data flow threads from imperative C programs

- direct automatic parallelization
- exploit additional parallelism at finer granularity within OpenStream tasks

Parallel intermediate representations

- bring parallel semantics into compiler representations
- avoid early lowering of parallel constructs to opaque runtime calls
- avoid losing sequential optimization opportunities
Extracting Coarse-Grain Data Flow Threads

- scalar case implemented in GCC
  - generalization of *Parallel-Stage Decoupled Software Pipelining* [E. Raman, G. Ottoni, A. Raman, M. J. Bridges, and D. I. August. Parallel-stage decoupled software pipelining. CGO 2008]
Extracting Coarse-Grain Data Flow Threads

- scalar case implemented in GCC
- generalization of Parallel-Stage Decoupled Software Pipelining [E. Raman, G. Ottoni, A. Raman, M. J. Bridges, and D. I. August. Parallel-stage decoupled software pipelining. CGO 2008]

```c
int foo () {
    S1 a0 = 0;
    S2 i0 = 0;
    S3 b0 = 0;
    S4 if (i0 <= 99) goto S5;
        else goto S10;
    S5 # i1 = phi(i0, i2);
    S6 a1 = i1;
    S7 b1 = bar (i1);
    S8 i2 = next (i1);
    S9 if (i2 <= 99) goto S5;
        else goto S10;
    S10 # a2 = $phi(a0, a1)
    S11 # b2 = $phi(b0, b1)
    S12 if (a2 > b2) goto S13;
        else goto S14;
    S13 # ret0 = a2;
    S14 ret1 = b2;
    S15 # ret2 = phi(ret0, ret1)
    S16 return ret;
}
```

1. Build the Program Dependence Graph under SSA
2. Typed fusion
3. Build the data flow PDG
Extracting Coarse-Grain Data Flow Threads

- scalar case implemented in GCC
  - generalization of Parallel-Stage Decoupled Software Pipelining [E. Raman, G. Ottoni, A. Raman, M. J. Bridges, and D. I. August. Parallel-stage decoupled software pipelining. CGO 2008]

- array/pointer case in the works
  - hybrid static/dynamic approach
Transfer objective: integration to OpenMP standard
Interest expressed by members of the OpenMP Architecture Review Board

Foster collaborations and technology transfers
IBM Haifa, U. of Sienna, Thales CS, Thales RT, CAPS Enterprise, Kalray, UPMC, BSC

- Open source implem. in GCC4.7
  ~30 kLoC
  GCC summit '08, '09

- New stream synchronization algorithms
  CASES'10, submitted EMSOFT'13

- Feed-Forward Data Flow
- Control-Driven Data Flow
  CDDF
  PhD thesis, submitted research report

- Proof techniques for concurrent algorithms
  PPoPP'13

- Benchmarks ~27 kLoC
  with third party contributions

- Profiling and visualiz. infrastr.
  collaboration with UPMC

OpenStream
http://openstream.info

- ACM TACO'13, IJPP'11, HiPEAC'11
- Automatic extraction of data flow threads from imperative programs (IEEE Micro'12)
Long term goal: efficient parallelization of irregular applications
Molecular dynamics, mesh refinement, particle transport...

Transfer objective: integration to OpenMP standard
Interest expressed by members of the OpenMP Architecture Review Board

Foster collaborations and technology transfers
IBM Haifa, U. of Sienna, Thales CS, Thales RT, CAPS Enterprise, Kalray, UPMC, BSC

Open source implem. in
GCC4.7
~30 kLoC
GCC summit '08, '09

New stream synchronization algorithms
CASES'10, submitted EMSOFT'13

Feed-Forward Data Flow

Control-Driven Data Flow
CDDF
PhD thesis, submitted research report

Proof techniques for concurrent algorithms
PPoPP'13

Benchmark
~27 kLoC
with third party contributions

Profiling and visualization infrastructure

OpenStream
http://openstream.info
ACM TACO'13, IJPP'11, HiPEAC'11

Automatic extraction of data flow threads from imperative programs (IEEE Micro'12)
Future Work
Future Work

Control-Driven Data Flow
Future Work

Control-Driven Data Flow

- relaxing the synchronous hypothesis
  - synchronous control program
  - asynchronous tasks
Future Work

Control-Driven Data Flow

- relaxing the synchronous hypothesis
  - synchronous control program
  - asynchronous tasks
- program verification
  - correct-by-construction synchronous specification (control program)
  - user specified task-level invariants
  - CDDF determinism guarantees
Future Work

Control-Driven Data Flow

- relaxing the synchronous hypothesis
  - synchronous control program
  - asynchronous tasks
- program verification
  - correct-by-construction synchronous specification (control program)
  - user specified task-level invariants
  - CDDF determinism guarantees
- formal semantics for X10
  - prove deadlock freedom on X10 clocks
Future Work

Compilation
Future Work

Compilation

- intermediate representations for parallel programs
  - links between SSA and streaming
Future Work

Compilation

• intermediate representations for parallel programs
  • links between SSA and streaming
• polyhedral compilation of streaming programs
  • affine stream access functions
  • compile-time task level optimizations

• OpenStream backend for polyhedral compilation
  • use precise dataflow analysis information to generate point-to-point dependences
  • links with polyhedral Kahn process networks
Future Work

Compilation

- intermediate representations for parallel programs
  - links between SSA and streaming
- polyhedral compilation of streaming programs
  - affine stream access functions
  - compile-time task level optimizations
- OpenStream backend for polyhedral compilation
  - use precise dataflow analysis information to generate point-to-point dependences
  - links with polyhedral Kahn process networks
Future Work

Runtime optimization
Future Work

Runtime optimization

- task placement for locality
Future Work

Runtime optimization

- task placement for locality
- adaptive scheduling, dynamic tiling
Future Work

Runtime optimization

- task placement for locality
- adaptive scheduling, dynamic tiling
- program semantics under relaxed memory consistency hypotheses
Future Work

Runtime optimization

• task placement for locality
• adaptive scheduling, dynamic tiling
• program semantics under relaxed memory consistency hypotheses
• runtime deadlock detection in presence of dynamic, speculative aggregation
Future Work

Distributed memory and heterogeneous platform execution

- Owner Writeable Memory (OWM)
  - coherence protocol
  - code-generation geared *Software Distributed Shared Memory*
  - explicit *cache/publish* operations
- locality/bandwidth optimization
Impact and Dissemination

1. Used in 4 partnership research projects:
   - ACOTES (FP6) – IBM Haifa
   - TERAFLUX (FP7 FET-IP) – University of Sienna, Thales CS, CAPS
   - PHARAON (FP7 STREP) – Thales RT
   - ManycoreLabs (Investissements d’avenir - BGLE) – Kalray

2. Used in 3 ongoing PhD theses (ÉNS, INRIA and UPMC)

3. Used in a parallel programming course project at Politecnico di Torino.

4. Used at École Normale Supérieure as a back-end target for parallel code generation from the synchronous language Heptagon.

5. Ongoing development at Barcelona Supercomputing Center for providing a performance and T* portability OpenStream back-end for StarSs (based on our proposed code generation algorithm).

6. Ongoing work to port OpenStream on Kalray MPPA.

7. Used for developing and evaluating communication channel synchronization algorithms by Preud’Homme et al. in “An Improvement of OpenMP Pipeline Parallelism with the BatchQueue Algorithm,” ICPADS 2012.


9. Source code publicly available on Sourceforge http://sourceforge.net/p/open-stream/