Automatic Discovery of Multi-level Parallelism in MATLAB

Arun Chauhan
Google Inc. and Indiana University

Pushkar Ratnalikar
Indiana University

February 7, 2015
Motivation
Multi-level Parallelism

Getting the premises right

- Algorithm-level parallelism
- Software-level parallelism
- Hardware-level parallelism
Multi-level Parallelism

Getting the premises right

Algorithm-level parallelism

Software-level parallelism

Hardware-level parallelism

- Novel parallel algorithms
- Specialized for GPUs
- Specialized for FPGAs

- Data parallelism
- Task parallelism

- Superscalar
- Out of order execution
- Speculative execution
- Branch prediction
Multi-level Parallelism

Getting the premises right

**Motivation**

**Data-flow**

**Approach**

**Experiments**

**Granularity**

**Conclusion**

---

**No standard tools**

- Algorithm-level parallelism

- Novel parallel algorithms
- Specialized for GPUs
- Specialized for FPGAs

**OpenMP, MPI, Intel TBB**

- Software-level parallelism
- Data parallelism
- Task parallelism

**Largely automatic, compilers**

- Hardware-level parallelism
- Superscalar
- Out of order execution
- Speculative execution
- Branch prediction
Multi-level Parallelism

Getting the premises right

**No standard tools**

- Algorithm-level parallelism

**OpenMP, MPI, Intel TBB**

- Software-level parallelism
  - Data parallelism
  - Task parallelism

**Largely automatic, compilers**

- Hardware-level parallelism
  - Superscalar
  - Out of order execution
  - Speculative execution
  - Branch prediction
Multi-level Parallelism in Software

Getting the premises right

Software-level parallelism
Multi-level Parallelism in Software

Getting the premises right

Statement-level parallelism

Loop-level (data) parallelism

Function-level (task) parallelism

Component-level parallelism

Software-level parallelism
Multi-level Parallelism in Software

Getting the premises right

Statement-level parallelism

Loop-level (data) parallelism

Function-level (task) parallelism

Component-level parallelism

- Multi-threaded built-in libraries
- Language constructs
  - E.g., parfor
- Parallel third-party libraries
  - E.g., GPUMat and StarP

Software-level parallelism
Parallelism in MATLAB

- ILP for free, as always
- Carefully optimized libraries
  - Multi-threaded (for data parallelism)
  - Highly tuned (to utilize machine vector instructions)
- Language-level constructs
  - Programmer identifies parallel loops
  - Programmer identifies parallel tasks
  - Programmer identifies GPU-bound statements
Parallelism in MATLAB

- ILP for free, as always
- Carefully optimized libraries
  - Multi-threaded (for data parallelism)
  - Highly tuned (to utilize machine vector instructions)
- Language-level constructs
  - Programmer identifies parallel loops
  - Programmer identifies parallel tasks
  - Programmer identifies GPU-bound statements

Reliance on programmers untenable
Parallelism in MATLAB

- ILP for free, as always
- Carefully optimized libraries
  - Multi-threaded (for data parallelism)
  - Highly tuned (to utilize machine vector instructions)
- Language-level constructs
  - Programmer identifies parallel loops
  - Programmer identifies parallel tasks
  - Programmer identifies GPU-bound statements

Wish to automate

Reliance on programmers untenable
Motivation

What is the Right Model of Parallelism?

- One that does not require programmers to write parallel code at all!

- But, at the system level:
  - Need to exploit parallelism at all levels of hardware and software
  - Need to match the parallelism in the application to the underlying hardware
Data-flow Model of Computing
Data-flow computing ...
Data-flow computing …

Already exists in hardware
Data-flow computing ...

Already exists in hardware
Data-flow computing ...

\[
\begin{align*}
  d & \leftarrow a + b \\
  e & \leftarrow b \times c \\
  f & \leftarrow d \div e \\
  g & \leftarrow d + e + c
\end{align*}
\]

Already exists in hardware.
Data-flow computing ...

\[
d \leftarrow a + b \\
e \leftarrow b \times c \\
f \leftarrow d \div e \\
g \leftarrow d + e + c
\]

Already exists in hardware
Can be described procedurally
Motivation

Data-flow

Approach

Experiments

Granularity

Conclusion

... at the Right Granularity

Macro Data-flow Computing

A ← A + B
E ← B × C
F ← D ÷ E
G ← D + E + C

• Each operation a *task* in a task-parallel library (Intel TBB)
• Low amortized creation and deletion cost
• The operation can be data-parallel (multi-threaded)
• The operation could be an optimized and parallelized library function
Data-flow Execution for MATLAB Programs

- Programmers do not need to think about it
- Great for legacy code
- Allows us to utilize the existing and already implemented modes of parallelism
- Makes use of the specialized libraries, incorporating specialized expert knowledge
- Has the potential to utilize all levels of parallelism afforded by modern hardware
Data-flow Execution for MATLAB Programs

- Programmers do not need to think about it
- Great for legacy code
- Allows us to utilize the existing and already implemented modes of parallelism
- Makes use of the specialized libraries, incorporating specialized expert knowledge
- Has the potential to utilize all levels of parallelism afforded by modern hardware

All we need is automatic extraction!
Challenges

- Right granularity for “operations”
- Memory
  - Keep the footprint in check
  - Minimize memory copies
- Programming
  - Automatically generate data-flow-style execution from procedural description
  - Respect all data- and control-dependencies
- Run-time
  - Schedule operations smartly
Approach
Approach

- **Granularity**
  - Treat each array statement as an atomic data-flow operation, replicate scalar operations liberally
  - Merge to coarsen the granularity without decreasing parallelism

- **Memory**
  - Scalars are free, arrays are mutable (hybrid data-flow / procedural)

- **Programming**
  - Compiler analysis to determine data and control dependencies
  - Tasks can call libraries or be implemented as explicit loops

- **Run-time**
  - Custom run-time around Intel Threading Building Blocks (TBB)
Utilizing Parallelism at Multiple Levels

- Across operations
  - Task parallelism (or statement-level parallelism)

- Within operations
  - Use multi-threaded library operations
  - Parallelize loops implied by array operations

- More parallelism …
  - We handle one user function at a time
Example: Array Statements

MATLAB Code

```
S_0  z = rand(n,n);
S_1  a = v + f;
S_2  b = x + y;
while (c)
    S_3  b(:,i) = a ./ pi;
    S_4  z = b + z;
end
S_5  V = z';
```
Example: Array Statements

**MATLAB Code**

\begin{align*}
S_0 & \quad z = \text{rand}(n,n); \\
S_1 & \quad a = v + f; \\
S_2 & \quad b = x + y; \\
\text{while (c)} & \\
    S_3 & \quad b(:,i) = a ./ pi; \\
\text{end} & \\
S_4 & \quad z = b + z; \\
S_5 & \quad V = z';
\end{align*}

**Data dependencies**

![Data dependence graph](image)
Example: Array Statements

MATLAB Code

```
S0  z = rand(n,n);
S1  a = v + f;
S2  b = x + y;
while (c)
    S3  b(:,i) = a ./ pi;
    S4  z = b + z;
end
S5  V = z';
```

Data dependencies

```
S2  S1  S0
S3  |   |   \
    v   v   v
S4  |   |   \
    v   v   v
S5
```

Static Data-flow graph

```
Begin_F
   /   \
  c   c
S2  S1  S0
   /   \
  c   c
S3  S4
   /   \
  c   c
S5
```

Motivation

Example: Array Statements

1. Categorize dependencies that cross the loop boundary. Two additional tasks have been added, which are described later. There are three cases of data flow that can occur with loops:
   - Data flowing from outside the loop into the loop body. In the Figure 1 diagram, this is represented by the edges pointing into the loop body. The dependency from the loop control variable `c` to `S3` indicates that `S3` depends on `c`.

2. Handling loops with unknown iteration counts (i.e., while-loops).
   - A while loop is shown in the diagram, with the condition `c` being checked at the top of the loop. The loop body includes operations on arrays `a`, `b`, and `z`, with dependencies indicated by edges.

3. Handling data dependencies crossing iteration boundaries, including loop-termination.
   - The loop terminates with the update of `V` using the transpose of `z`.

Approach

The compiler and runtime system cooperate to manage the creation, execution, and destruction of tasks such that the control and data-dependencies of the problem are honored. In this section, we discuss the design of each major component.

Experiments

Handing data dependencies crossing iteration boundaries, including loop-termination.

Conclusion

The approach aims to avoid the costs associated with translating high-level loop constructs to lower-level parallel constructs. It involves solving two main problems:

- Handling data dependencies crossing iteration boundaries, including loop-termination.
- Handling loops with unknown iteration counts (i.e., while-loops).
Example: Array Statements (Modified)

\[ S_0 \ f = \text{rand}(n,n); \]
\[ S_1 \ a = v + f; \]
\[ S_2 \ b = x + y; \]
\[ \text{while} \ (c) \]
\[ \ \ \ \ S_3 \ b(:,i) = a ./ \pi; \]
\[ \ \ \ \ S_4 \ z = d + z; \]
\[ \text{end} \]
\[ S_5 \ V = z'; \]
Example: Array and Scalar Statements

```matlab
n = length(v);
k = 500;
H = zeros(k,k);
V = zeros(n,k);
...
...
j = 2;
tmp4 = j <= k;
while(tmp4),
  ...
  ...
  V(:,j) = v;
  H(1:j,j) = h;
  j = j + 1;
tmp4 = j <= k;
end
```
Example: Array and Scalar Statements

```matlab
n = length(v);
k = 500;
H = zeros(k,k);
V = zeros(n,k);
...
...
j = 2;
tmp4 = j <= k;
while(tmp4),
...
...
V(:,j) = v;
H(1:j,j) = h;
j = j + 1;
tmp4 = j <= k;
end
```
Example: Array and Scalar Statements

```matlab
n = length(v);
k = 500;
H = zeros(k,k);
V = zeros(n,k);
... ...
j = 2;
tmp4 = j <= k;
while(tmp4),
...
V(:,j) = v;
H(1:j,j) = h;
j = j + 1;
tmp4 = j <= k;
end
```

```matlab
k$1 = 500;
H$1= zeros(k$1,k$1);
j$1 = 2;
tmp4$1= j$1<=k$1;
```

```
n$1 = length(v$0);
k$1 = 500;
V$1 = zeros(n$1,k$1);
j$1 = 2;
tmp4$1= j$1<=k$1;
v$1(v$1_data);
V$1(V$1_data);
```

```
V$1(:,j$2) = v$1;
j$3 = j$2+1;
tmp4$3=j$3<=k$1;
```

```
H$1(1:j$2,j$2)=h$1;
j$3 = j$2+1;
tmp4$3=j$3<=k$1;
```
Example: Array and Scalar Statements

n = length(v);
k = 500;
H = zeros(k,k);
V = zeros(n,k);
...
...
j = 2;
tmp4 = j <= k;
while(tmp4),
...
V(:,j) = v;
H(1:j,j) = h;
j = j + 1;
tmp4 = j <= k;
end

k$1 = 500;
H$1= zeros(k$1,k$1);
j$1 = 2;
tmp4$1= j$1<=k$1;
n$1 = length(v$0);
k$1 = 500;
V$1 = zeros(n$1,k$1);
j$1 = 2;
tmp4$1= j$1<=k$1;
v$1(v$1_data);
V$1(:,j$2) = v$1;
j$3 = j$2+1;
tmp4$3= j$3<=k$1;
H$1(1:j$2,j$2)=h$1;
j$3 = j$2+1;
tmp4$3= j$3<=k$1;
**Example: Array and Scalar Statements**

```matlab
n = length(v);
k = 500;
H = zeros(k,k);
V = zeros(n,k);
...
j = 2;
tmp4 = j <= k;
while(tmp4),
  ...
  V(:,j) = v;
  H(1:j,j) = h;
  j = j + 1;
tmp4 = j <= k;
end
```

```
k$1 = 500;
H$1 = zeros(k$1,k$1);
j$1 = 2;
tmp4$1 = j$1 <= k$1;

n$1 = length(v$0);
k$1 = 500;
V$1 = zeros(n$1,k$1);
j$1 = 2;
tmp4$1 = j$1 <= k$1;

V$1(:,j$2) = v$1;
j$3 = j$2 + 1;
tmp4$3 = j$3 <= k$1;

H$1(1:j$2,j$2) = h$1;
j$3 = j$2 + 1;
tmp4$3 = j$3 <= k$1;
```

```
tmp4$1
...
```

**Static Data-flow Graph**

Figure 2 gives the high-level overview of the translation. The code in Fig-
Example: Accounting for Control Flow
Without any extra controller tasks

\begin{verbatim}
2 Fx$1 = zeros(n$0, a$0);
3 drx$1 = zeros(n$0, n$0);
4 x$1 = Fx$1(:, n$0);
5 G$1 = 1e-11;
6 t$1 = 1;
7 tmp1$1 = t$1 <= T$0;
8 while(tmp1$2)
9  k$2 = 1;
10  tmp2$2 = k$2 <= n$0;
11  while(tmp2$3)
12    j$3 = 1;
13    tmp3$3 = j$3 <= n$0;
14    while(tmp3$4)
15      Fx$5(:, k$3) = G$1;
16      j$5 = j$4 + 1;
17      tmp3$5 = j$5 <= n$0;
18    end
19  k$4 = k$3 + 1;
20  tmp2$4 = k$4 <= n$0;
21  end
22 tmp4$2 = t$2 == 2;
23 if(tmp4$2);
24  continue;
25 end
26 Fx$6(:, t) = G$1 * drx$1;
27 f$1 = Fx$6(:, k$3);
28 t$3 = t$2 + dT$0;
29 tmp1$3 = t$3 <= T$0;
30 end
\end{verbatim}

Control dependence graph

Automatic Discovery of Multi-level Parallelism in MATLAB
Computing the Edge Conditions

1 Algorithm: ComputeDepConditions
2 Input: CDG $G$, Source $src$, Destination $dst$, CFG $cfg$
3 Output: Predicate Expression $L$
4 $S \leftarrow \{c_1, \ldots, c_k, s_1, \ldots, s_k\} /\ast$ seq. of all cond. exprs enclosing $src$ /\ast$
5 $D \leftarrow \{c_1, \ldots, c_k, d_1, \ldots, d_k\} /\ast$ seq. of all cond. exprs enclosing $dst$ /\ast$
6 $L \leftarrow \neg(s_1 \land, \ldots, \land s_k) \land (c_1 \land, \ldots, \land c_k)$
7 for each $n$ in $\{c_1, \ldots, c_k\}$ do
8     if $(c \leftarrow ClearPath(src, n, dst, cfg))$ then
9         $L \leftarrow L \land c$
10     else
11         break;
Some Implementation Details

- Intel Threading Building Blocks (TBB) for tasks
- Task types
  - Subclass tbb::task
  - A type for each operation
- Concurrent hash-map for waiting tasks
  - Created, but waiting for input
  - Removed as soon as start running
- Atomic counters to track ready inputs
- Armadillo matrix library
  - Readable C++ syntax
  - Efficient implementation with expression templates
Automatic Discovery of Multi-level Parallelism in MATLAB

Arun Chauhan, Google Inc. and Indiana University
achauhan@cs.indiana.edu

Pushkar Ratnalikar
pratnali@cs.indiana.edu

Abstract
The popularity of MATLAB in scientific and engineering domains is tempered by its performance. Highly optimized libraries, automatic thread-level parallelism for large computations within libraries, and loop-level parallel constructs in the language are some of the ways in which the language implementers have tried to recoup the performance. Greater potential for parallelism exists in typical MATLAB programs that remains unexploited. We discuss our MathWorks-sponsored effort in automatically exploiting parallelism in MATLAB programs using a combination of compile-time and run-time techniques. Our approach is inspired by data-flow-style computation and makes use of some modern C++ libraries for generating highly readable code with support for data parallelism and GPUs.

1 Motivation and Design
Computing on modern high performance machines afford parallelism at multiple levels, from the vector instructions on a single core to multiple multi-core nodes connected through fast interconnects. Graphical Processing Units (GPUs) add heterogeneity and scheduling complexity to the mix.

In order to shield non-expert users from the complexities and interactions of the various forms of parallelism it is often wrapped inside libraries. The libraries are carefully optimized to make use of the vector instructions of the underlying hardware, to use multiple threads when the amount of computation makes it worthwhile, and to provide versions that might utilize accelerators, such as GPUs. Indeed, this is the dominant approach to parallelism in MATLAB. With a few exceptions, such as having to specify the computations to be performed on GPUs, the process is largely automatic for the users and, hence, highly attractive from the perspective of programmability.

However, it suffers from two major inefficiencies: the decisions about parallelism must be made inside libraries, which are only locally optimal, at best; and parallelism across library functions is hard to exploit.

Our primary motivation behind this work is to eliminate these inefficiencies without burdening the user with additional code annotations, such as those required for using MATLAB parallel constructs. The high-level nature of MATLAB makes code analysis sufficiently accurate in the common cases that the compiler is able to expose the parallelism that MATLAB libraries would be unable to exploit and which would be non-trivial to express with the repertoire of MATLAB’s parallel constructs. In order to make full use of the parallelism we emit C++ code, instead of MATLAB, which lets us use a custom run-time system combined with modern C++ libraries, such as Intel Threading Building Blocks (TBB) for task scheduling, Armadillo for data-parallel matrix operations, and Thrust or ArrayFire for GPUs.

The MATLAB libraries continue to be available to the translated code. However, their lack of reentrance property prevents us from making concurrent calls to any single MATLAB library function. Figure 1 shows the overall system.

Our computation model is inspired by coarse-grained data-flow computation [3], which is also implicit in streaming applications and has been used in large practical systems, such as MillWheel at Google [1]. The wide applicability of the model makes it a powerful mechanism to exploit parallelism at multiple levels and scales, including heterogenous parallelism with GPUs.

Empirical Evaluation
Speedups

WaveCrossCov

Input size

Speedup over MATLAB

- Data-Parallel
- (Task+Data)-Parallel
Automatic Discovery of Multi-level Parallelism in MATLAB

Arun Chauhan, Google Inc. and Indiana University
**Speedups**

**WaveCrossCov**

- Data-Parallel
- (Task+Data)-Parallel

**Adi**

- Data-Parallel
- (Task+Data)-Parallel

**Gaussr**

- Data-Parallel
- (Task+Data)-Parallel
- (Task+Data)-Parallel coarse

**Input size**

- 0
- 5
- 10
- 15
- 20
- 25
- 50
- 100
- 150

**Speedup over MATLAB**

- 0
- 2
- 4
- 6
- 8
- 10
- 12

**Automatic Discovery of Multi-level Parallelism in MATLAB**

Arun Chauhan, Google Inc. and Indiana University
Automatic Discovery of Multi-level Parallelism in MATLAB

Arun Chauhan, Google Inc. and Indiana University
Experimental Setup

- Dual 16-core AMD Opteron 6380
  - 2.5 GHz, 64 GB DDR3 memory, 16 MB L3 cache
- Cray Linux Environment 4.1.UP01
- GCC 4.8.1
- Armadillo C++ library version 4.000
- Intel MKL 11.0
- Median of 10 runs
- Studied several benchmarks, only some reported
  - Studied code with large proportion of array operations
Performance and Concurrency

Motivation

Data-flow

Approach

Experiments

Granularity

Conclusion

Performance and Concurrency

Speedup over MATLAB

Input size

WaveCrossCov

Data-Parallel
(Task+Data)-Parallel

Wav1

Data-Parallel
(Task+Data)-Parallel

Gaussr

Data-Parallel
(Task+Data)-Parallel
(Task+Data)-Parallel coarse
Performance and Concurrency

Motivation

Data-flow

Approach

Experiments

Granularity

Conclusion

Performance and Concurrency

Input size

Speedup over MATLAB

Data-Parallel
(Task+Data)-Parallel

WaveCrossCov

Input size

Wav1

Input size

Gaussr

Input size

Num. of concurrent tasks

WaveCrossCov

Input size

Wav1

Input size

Gaussr

Input size

Automatic Discovery of Multi-level Parallelism in MATLAB

Arun Chauhan, Google Inc. and Indiana University
Performance and Concurrency (contd.)

**NBody 3D**
- Speedup over MATLAB
  - Data-Parallel
  - (Task+Data)-Parallel

**Arnoldi**
- Speedup over MATLAB
  - Data-Parallel
  - (Task+Data)-Parallel

**Num. of concurrent tasks**
- NBody 3D
- Arnoldi
Task Efficiency on 16 Cores

Task Efficiency

Average task efficiency

WaveCrossCov, Wav1, Gaussr, NBody 1D, Adi, QR, Burgpr, Heated Plate, Compute New, NBody 3D, Arnoldi

Automatic Discovery of Multi-level Parallelism in MATLAB

Arun Chauhan, Google Inc. and Indiana University
Task Efficiency on 16 Cores

Can we use this information?
Granularity Adjustment
Task Granularity can have Dramatic Impact

Motivation
Data-flow
Approach
Experiments
Granularity
Conclusion

Granularity

Task Granularity can have Dramatic Impact
Challenges and Opportunities

Problems

- Cost model for when and how much to coarsen
- Challenging to estimate the gains
- Should not sacrifice parallelism (not too much)

Potential gains

- Reduced task creation and deletion overhead
- Improved data locality
- Also possible to fuse loops and scalarize array temporaries
Cases to Consider

(a) $x \rightarrow y \rightarrow z$

(b) $x \rightarrow y \rightarrow z$

(c) $x \rightarrow y \rightarrow z$

(d) $x \rightarrow y1 \rightarrow y2 \rightarrow z$
Properties

- No dependency violation
- No reduction in parallelism
- Prefer merging related tasks for improved locality
Example: GaussR

G = 1e-11;
drx = Rx-Rx(k);
dry = Ry-Ry(k);
drz = Rz-Rz(k);
r_tmp1 = drx.*drx;
r_tmp2 = dry.*dry;
r_tmp3 = drz.*drz;
r = r_tmp1 + r_tmp2 + r_tmp3;
r(k) = 1.0;
M = m * m(k);
M(k) = 0.0;
f = G*(M./r);
r = sqrt(r);
frx = f.*drx;
fry = f.*dry;
frz = f.*drz;
Fx(k) = mean(frx)*n;
Fy(k) = mean(fry)*n;
Fz(k) = mean(frz)*n;

Static data-flow graph (21 nodes)

Coarsened graph (10 nodes)
Concluding Remarks
Take-away Message

- We use data-flow style of parallelism to be able to extract parallelism at all levels, automatically, from MATLAB
- We can extract parallelism that the libraries cannot utilize
- We utilize and build upon the existing modes of parallelism, instead of discarding them
- We can utilize software tools to create loop-level parallelism (e.g., using OpenMP)
- We would like to do better!
http://www.cs.indiana.edu/~achauhan