DATA TYPES FOR PARALLEL PROGRAMMING AND AUTOTUNING

David Padua

Department of Computer Science
University of Illinois at Urbana-Champaign
Ganesh Bikshandi
James Brodman
Carl Evans
Jia Guo
Dan Hoeflinger
Brandon Moore
Christoph von Praun
George Almasi
Diego Andrade
Basilio Fraguela
Maria Garzaran
Introduction:
The parallel programming problem
The challenge of our times

Widespread parallel programming → a dip in productivity.

Our quest in software research is (and has always been): **improve productivity**. In the near term → try to recover from the setback.
What to do?

- Languages/Notation
- Tools (compilers/autotuning /libraries)
- Education.
Focus of this talk
Programming notation

- Prefer no new languages
- Focus on data types
  - Can be used on its own or in combination with control parallelism
- Extensions for autotuning
  - Second best way to deploy autotuning after library generation
Arrays and Data Parallel Programming
Array Languages

- Popular among scientists and engineers.
  - Fortran 90 and successors
  - MATLAB
- Parallelism not the reason for this notation.
Array Languages

- Convenient notation for linear algebra and other algorithms
  - More compact
  - Higher level of abstraction

```plaintext
C = A + B
S += sum(A)
```

```plaintext
do i=1,n
  do j=1,n
    C(i,j) = A(i,j) + B(i,j)
  end do
end do
```
Arrays for parallel programming

- Used in the past: Illiac IV, Connection machine, ...
- Google’s MapReduce
- Intel’s Ct.
  - Surprisingly, not universally used.
  - Vectorization has been used instead of explicit vector operations.
  - Fortran 90 is translated into loops and later vectorized
How well does vectorization work?

Benefits of using arrays for parallel programming

- Data parallel programs based on arrays resemble conventional, serial programs.
  - Parallelism is **encapsulated**.
  - Parallelism is **structured**.
  - Composable
  - Portable
  - Can run on any class of machine for which the appropriate operators are implemented

- Operations implemented as *parallel loops in shared memory*
- Operations implemented as *messages if distributed memory*
- Operations implemented with *vector intrinsics for SIMD*
Benefits of using arrays for parallel programming

- Determinacy can be enforced or non-determinacy encapsulated
- Autotuning/compiling is facilitated by the higher level notation:
  - ATLAS from MATLAB
  - Data structure selection
- Seems that many autotuning features can be included in declaration.
Tiles
A fundamental idea in computing

Organizing Matrices and Matrix Operations for Paged Memory Systems

A. C. McKellar and E. G. Coffman, Jr.
Princeton University, Princeton, New Jersey

usually expressed in units of machine instructions, memory references, etc., depending on what is most convenient.

If for a given page replacement algorithm a program can be designed to minimize page faulting, then, other things being equal, mean page residence times are maximized.

How to minimize page faulting for the very important class of problems involving the basic operations of matrix algebra is investigated. To this end, we examine methods of representing (paginating) matrices and sequencing arithmetic operations over these representations to yield
Wide range of applicability

- Locality
- Avoiding contention on centralized data structures
- Data distribution
- Load balancing
Tiled Arrays

- Arrays where tiles are
  - Referenced explicitly.
  - Manipulated using array operations such as reductions, gather, etc..

Tiled Arrays

Distributed
Locality
Locality
Array operations

- `repmat(h, [1, 3])`
- `circshift(h, [0, -1])`
- `transpose(h)`
Higher Level Operations

- Well-known
  - reduce, circular shift, replicate, transpose, etc

- Programmers create new complex parallel operators with **hmap**
  - Applies user defined operators to each tile of the HTA
    - And corresponding tiles if multiple HTAs are passed as input
  - Application of operator occurs in parallel across tiles
### Cannon's Matrix Multiplication

<table>
<thead>
<tr>
<th></th>
<th>A(_{00})</th>
<th>A(_{01})</th>
<th>A(_{02})</th>
</tr>
</thead>
<tbody>
<tr>
<td>A(_{10})</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A(_{20})</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>B(_{00})</th>
<th>B(_{01})</th>
<th>B(_{02})</th>
</tr>
</thead>
<tbody>
<tr>
<td>B(_{10})</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B(_{20})</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Initial Skew

#### Shift-Multiply-Add
function c = matrix_mult(a,b,n)
    %Main loop
    for i = 1:n
        c = c + a * b;
        a = circshift( a, [0, -1] );
        b = circshift( b, [-1, 0] );
    end
end
FT

(a)

(b)

\[ u = \text{fft} \ (u, \ [], 1); \]
\[ u = \text{fft} \ (u, \ [], 2); \]
\[ u = \text{dpermute} \ (u, [3 \ 1 \ 2]); \]
\[ u = \text{fft} \ (u, \ [], 1); \]
Tuning: Number of tiles, tile size

Example: Sameh’s Spike algorithm

\[ A = \text{diag}(A_1, \ldots, A_p) \cdot W \]

Reduced system

Retrieve solution

Optimization parameter part of data type
\[ A = \text{hta}( \ldots \text{tile\_shape}=[n?,n?] \ldots) \]
A compact, readable notation

Lines of Code. HTA vs. MPI
User Defined Operations - Merge

```java
Merge (HTA input1, HTA input2, HTA output) {
    i = input1.size() / 2
    input1.addPartition(i)

    j = h2.location_first_gt(input1[i])
    input2.addPartition(j)

    k = i + j
    output.addPartition(k)

    hmap(Merge(), input1, input2, output)
}
```
Tuning: When to switch to serial merge

Optimization parameter part of data type

```c
Merge (HTA[n?] input1, HTA[n?] input2, HTA output ) {
    ... serial merge ...
}
```

Size $\leq n$?
SLOC comparison: HTAs vs. TBBs

<table>
<thead>
<tr>
<th>Code</th>
<th>Lines (HTA)</th>
<th>Lines (TBB)</th>
<th>HTA reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average</td>
<td>28</td>
<td>39</td>
<td>+28%</td>
</tr>
<tr>
<td>Seismic</td>
<td>304</td>
<td>295</td>
<td>-3%</td>
</tr>
<tr>
<td>Parallel merge</td>
<td>70</td>
<td>74</td>
<td>+5.4%</td>
</tr>
<tr>
<td>Game of life</td>
<td>97</td>
<td>309</td>
<td>+69%</td>
</tr>
<tr>
<td>Substring finder</td>
<td>49</td>
<td>49</td>
<td>0%</td>
</tr>
</tbody>
</table>
**Dynamic tiling**

**Algorithm:** \( [A, p] := LUPIV_BLK( A ) \)

**Partition**

\[
A \rightarrow \begin{array}{c|c}
A_{TL} & A_{TR} \\
\hline
A_{BL} & A_{BR}
\end{array}
, \quad p \rightarrow \begin{array}{c}
p_T \\
\hline
p_B
\end{array}
\]

where \( A_{TL} \) is \( 0 \times 0 \), \( p_T \) has 0 elements

while \( n(A_{TL}) < n(A) \) do

**Determine block size** \( b \)

\[
\begin{array}{c|c|c}
A_{TL} & A_{TR} \\
\hline
A_{BL} & A_{BR}
\end{array} \rightarrow \begin{array}{c|c|c}
A_{00} & A_{01} & A_{02} \\
\hline
A_{10} & A_{11} & A_{12} \\
A_{20} & A_{21} & A_{22}
\end{array}
, \\
\begin{array}{c}
p_T \\
\hline
p_B
\end{array} \rightarrow \begin{array}{c}
p_0 \\
\hline
p_1 \\
\hline
p_2
\end{array}
\]

where \( A_{11} \) is \( b \times b \), \( p_1 \) is \( b \times 1 \)

\[
\begin{align*}
\frac{A_{11}}{A_{21}}, & \quad p_1 := LUPIV \frac{A_{11}}{A_{21}} \\
\frac{A_{10}}{A_{20}}, & \quad p_2 := P(p_1) \frac{A_{10}}{A_{20}} \frac{A_{12}}{A_{22}} \\
A_{12} := L_{11}^{-1} A_{12} \\
A_{22} := A_{22} - A_{21} A_{12}
\end{align*}
\]

Continue with

\[
\begin{array}{c|c|c}
A_{TL} & A_{TR} \\
\hline
A_{BL} & A_{BR}
\end{array} \rightarrow \begin{array}{c|c|c}
A_{00} & A_{01} & A_{02} \\
\hline
A_{10} & A_{11} & A_{12} \\
A_{20} & A_{21} & A_{22}
\end{array}
, \\
\begin{array}{c}
p_T \\
\hline
p_B
\end{array} \rightarrow \frac{\begin{array}{c}
p_0 \\
\hline
p_1 \\
\hline
p_2
\end{array}}{p_2}
\]

endwhile

```c
void lu(HTA<double,2> A, HTA<int,1> p,int nb) {
    A.part((0,0),(0,0));
    p.part((0), (0));
    while(A(0,0).lsize(1)<A.lsize(1)){
        int b = min(A(1,1).lsize(0), nb);
        A.part((1,1),(b,b));
        p.part((1),  (b));
        dgetrf(A(1:2,1), p(1));
        dlaswp(A(1:2,0), p(1));
        dlaswp(A(1:2,2), p(1));
        trsm(HtaRight, HtaUpper, HtaNoTrans,
            HtaUnit, One, A(1,1),A(1,2));
        gemm(HtaNoTrans, HtaNoTrans, MinusOne,
            A(2,1), A(1,2), One, A(2,2));
        A.rmPart((1,1));
        p.rmPart((1));
    }
}
```
Data parallel operations on sets
Data Parallel Search Algorithms
Data Parallel Search Algorithms

Set \textit{work\_list}
Set \textit{states}
Set \textit{successors}

\textbf{Search( initial\_state )}
\hspace{1em} \textit{work\_list}.add( initial\_state )

while ( not done )
\hspace{1em} \text{map( SELECT(), work\_list, states )}

if ( not found\_solution(\textit{work\_list} )
\hspace{1em} \text{map( EXPAND(), states, successors )}
\hspace{1em} \text{map( UPDATE(), work\_list, successors )}
Tiling Sets

- To specify the tiling structure of a Tiled Set requires:
  - The number of tiles
  - A **mapping function** that takes an element of the set and specifies the destination tile
Tuning: mapping function

\[ A = hta( \ldots \text{mapping\_function} = \{dp-1, dp-2\} \ldots) \]
The End
struct PMergeRange {
    It begin1, end1; // [begin1,end1) 1st sequence to merge
    It begin2, end2; // [begin2,end2) 2nd sequence to merge
    It out;       // where to put merged sequence
    bool empty() const { return (end1-begin1)+(end2-begin2)==0; }
    bool is_divisible() const { return (end1 - begin1) > GRAINSIZE; }
    PMergeRange( PMergeRange& r, split ) {
        begin1 = r.begin1 + (r.end1 - r.begin1)/2;
        begin2 = lower_bound(r.begin2, r.end2, *begin1);
        out = r.out + (begin1 - r.begin1) + (begin2 - r.begin2);
        r.end1 = begin1;
        r.end2 = begin2;
    }
    PMergeRange(It b1, It e1, It b2, It e2, It out1) {
        begin1(b1), end1(e1), begin2(b2),end2(e2), out(out1)
    }
};

struct PMergeBody {
    void operator() (PMergeRange& r) const {
        /* sequential merge */
    }
};

struct PMerge {
    void operator() (HTA1 out, HTA1 in1, HTA1 in2) {
        int in1_size = in1.lsize(0);
        if (in1_size > GRAINSIZE) {
            /* parallel merge */
            int midpos_in1 = in1_size / 2;
            int cutpoint_in2 = in2.lowerBoundPos(in1[midpos_in1]);
            in1.part([0], [midpos_in1]);
            in2.part([0], [cutpoint_in2]);
            out.part([0], [midpos_in1 + cutpoint_in2]);
            out.hmap(PMerge(), in1, in2);
            in1.rmPart();
            in2.rmPart();
            out.rmPart();
        } else {
            /* sequential merge */
        }
    }
};

... out.hmap(Pmerge(), in1, in2);

... parallel_for(PMergeRange(begin1, end1, begin2, end2, out),
                 PMergeBody());
# Performance vs. TBB

<table>
<thead>
<tr>
<th>Code</th>
<th>HTA</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th>TBB</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>8</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Average</td>
<td>5.2</td>
<td>2.5</td>
<td>2.1</td>
<td>2.2</td>
<td>1.5</td>
<td>3.1</td>
<td>2.5</td>
<td>2.2</td>
<td>2.4</td>
<td>2.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Seismic</td>
<td>8133.8</td>
<td>4234.1</td>
<td>2937.6</td>
<td>2399.8</td>
<td>1577.9</td>
<td>5975.2</td>
<td>3117.8</td>
<td>2328.4</td>
<td>1853.2</td>
<td>1458.7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Parallel merge</td>
<td>68.6</td>
<td>36.4</td>
<td>34</td>
<td>22.2</td>
<td>21.3</td>
<td>73</td>
<td>36.1</td>
<td>26</td>
<td>20.7</td>
<td>19.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Game of life</td>
<td>4957</td>
<td>2465</td>
<td>2577.4</td>
<td>1745.7</td>
<td>1088.1</td>
<td>4473.9</td>
<td>2745.5</td>
<td>2130.2</td>
<td>1813.3</td>
<td>1381.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Substring finder</td>
<td>5885.9</td>
<td>2992</td>
<td>2003.7</td>
<td>1541.6</td>
<td>768.9</td>
<td>6380.2</td>
<td>3203.8</td>
<td>2132.1</td>
<td>1610</td>
<td>820.3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*TIMES IN MILISECONDS, FOR THE TBB AND HTA VERSIONS IN THE TWO QUAD-CORE XEON SERVER USING 1,2,3,4 AND 8 CORES RESPECTIVELY*

<table>
<thead>
<tr>
<th>Code</th>
<th>HTA</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th>TBB</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>12</td>
<td>16</td>
<td></td>
<td></td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>Average</td>
<td>25.1</td>
<td>11.2</td>
<td>7.2</td>
<td>4.5</td>
<td>3.8</td>
<td>3.5</td>
<td>23.8</td>
<td>13.1</td>
<td>11.4</td>
<td>11.2</td>
<td>11.7</td>
<td>11.3</td>
</tr>
<tr>
<td>Seismic</td>
<td>19359.5</td>
<td>11201.7</td>
<td>7503.81</td>
<td>4916.8</td>
<td>3712.2</td>
<td>4129.8</td>
<td>15552</td>
<td>8824.3</td>
<td>6124.7</td>
<td>4000.6</td>
<td>3748.6</td>
<td>3215.6</td>
</tr>
<tr>
<td>Parallel merge</td>
<td>199.2</td>
<td>128.3</td>
<td>79.6</td>
<td>52.1</td>
<td>44.8</td>
<td>44.5</td>
<td>202.4</td>
<td>116.7</td>
<td>66.9</td>
<td>44.3</td>
<td>38.1</td>
<td>35</td>
</tr>
<tr>
<td>Game of life</td>
<td>19396.7</td>
<td>9486.7</td>
<td>6953</td>
<td>3478.9</td>
<td>2109.4</td>
<td>1690.8</td>
<td>16483.5</td>
<td>9623.1</td>
<td>6147.55</td>
<td>4.49</td>
<td>3654.7</td>
<td>3409.9</td>
</tr>
<tr>
<td>Substring finder</td>
<td>9510.4</td>
<td>4895.6</td>
<td>2455.3</td>
<td>1256.8</td>
<td>791.5</td>
<td>689.9</td>
<td>10689.4</td>
<td>5361.9</td>
<td>2692.9</td>
<td>1366.2</td>
<td>924.4</td>
<td>717</td>
</tr>
</tbody>
</table>

*TIMES IN MILISECONDS, FOR THE TBB AND HTA VERSIONS IN THE EIGHT DUAL-CORE ITANIUM 2 SERVER USING 1,2,4,8,12 AND 16 CORES RESPECTIVELY*