Shared Memory Parallelism in Ada: Load Balancing by Work Stealing

Jan Verschelde

University of Illinois at Chicago
Department of Mathematics, Statistics, and Computer Science
http://www.math.uic.edu/~jan
janv@uic.edu

www.phcpack.org

Ada devroom, FOSDEM 2018, 3 February, Brussels, Belgium
Outline

1 Problem Statement
- computing the permanent of a matrix
- high level parallel programming

2 Multitasking in Ada
- launching a crew of workers
- work stealing with multitasking
- application to polynomial system solving
1. **Problem Statement**
   - computing the permanent of a matrix
   - high level parallel programming

2. **Multitasking in Ada**
   - launching a crew of workers
   - work stealing with multitasking
   - application to polynomial system solving
all perfect matchings in a bipartite graph

Consider the adjacency matrix of a bipartite graph:

\[
A = \begin{bmatrix}
0 & 1 & 0 & 1 \\
1 & 1 & 1 & 0 \\
0 & 1 & 1 & 1 \\
1 & 0 & 1 & 0 \\
\end{bmatrix}
\]

\(\text{perm}(A) = 5\)

The permanent counts all perfect matchings in the graph:
row expansions

\[ A = \begin{bmatrix}
0 & 1 & 0 & 1 \\
1 & 1 & 1 & 0 \\
0 & 1 & 1 & 1 \\
1 & 0 & 1 & 0
\end{bmatrix} \]

We expand along the rows:

\[
\text{perm}(A) = 1 \times \begin{bmatrix}
1 & 1 & 0 \\
0 & 1 & 1 \\
1 & 1 & 0
\end{bmatrix} + 1 \times \begin{bmatrix}
1 & 1 & 1 \\
0 & 1 & 1 \\
1 & 0 & 1
\end{bmatrix}
\]

\[
= 1 \times \left( 1 \times \begin{bmatrix}
1 & 1 \\
1 & 0
\end{bmatrix} + 1 \times \begin{bmatrix}
0 & 1 \\
1 & 0
\end{bmatrix} \right)
\]

\[
+ 1 \times \left( 1 \times \begin{bmatrix}
1 & 1 \\
0 & 1
\end{bmatrix} + 1 \times \begin{bmatrix}
0 & 1 \\
1 & 1
\end{bmatrix} + 1 \times \begin{bmatrix}
0 & 1 \\
1 & 0
\end{bmatrix} \right)
\]

\[ = \ldots \]
computational experiments

The permanent of an \( n \)-by-\( n \) matrix \( A \) is

\[
\text{perm}(A) = \sum_{\sigma \in S_n} \prod_{i=1}^{n} a_{i,\sigma(i)},
\]

where \( S_n \) is the set of all permutations of \( n \) numbers, \( \#S_n = n! \).

On a MacBook Pro 3.1 GHz Intel Core i7, timings on randomly generated Boolean matrices, of dimension \( n = 14, 15, 16, 17 \), the CPU time in seconds:

<table>
<thead>
<tr>
<th>( n )</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>1.439</td>
</tr>
<tr>
<td>15</td>
<td>10.419</td>
</tr>
<tr>
<td>16</td>
<td>58.497</td>
</tr>
<tr>
<td>17</td>
<td>170.828</td>
</tr>
</tbody>
</table>
expanding the first two rows

Consider the first two rows in the matrix $A$:

$$A = \begin{bmatrix}
0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\
1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\
1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\
1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\
1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\
1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0
\end{bmatrix}$$

At the right are the expansions of the first two rows. Those expansions represent 11 computationally independent jobs.
1 Problem Statement
- computing the permanent of a matrix
- high level parallel programming

2 Multitasking in Ada
- launching a crew of workers
- work stealing with multitasking
- application to polynomial system solving
shared memory parallel programming

Consider a parallel computation by \( p \) processors:
1. all processors share the same memory space;
2. the jobs can be computed independently.

We can work with one static queue of jobs:
- The queue is initialized with jobs.
- Jobs are popped from the front of the queue.
- Popping jobs is guarded by a semaphore.
- Idle workers pop jobs till the queue is empty.

This is the work crew model of multithreading.
load balancing by work stealing

In the work crew model, processors take jobs from one queue.

In work stealing, underutilized processors steal jobs:

- Every processor has its own dequeue of jobs.
  A dequeue is a double ended queue, with beginning and end.

- Jobs are appended to the end of the dequeue.

  A processor treats its own dequeue as a stack:
  - pushing new jobs to the end,
  - popping jobs from the end.

- Processors with empty job queues steal jobs from others, popping from the beginning of their dequeue.

This idea appeared first in [Burton and Sleep, 1981]. The first provably good work stealing scheduling algorithm appeared in [Blumofe and Leiserson, 1994].
1 Problem Statement
- computing the permanent of a matrix
- high level parallel programming

2 Multitasking in Ada
- launching a crew of workers
- work stealing with multitasking
- application to polynomial system solving
procedure Workers is instantiated with a Job procedure, executing code based on the id number.

procedure Workers ( n : in natural ) is
  task type Worker ( id,n : natural );
  task body Worker is
  begin
    Job(id,n);
  end Worker;
procedure Launch_Workers ( i,n : in natural ) is
  w : Worker(i,n);
  begin
    if i < n
      then Launch_Workers(i+1,n);
    end if;
  end Launch_Workers;
begin
  Launch_Workers(1,n);
end Workers;
managing the job queue for the work crew

On input is a list of partially selected column indices. The job queue is then the corresponding list of pointers: each job requires the application of recursive row expansions.

The permanent computation is then a pleasingly parallel computation: no communication overhead during the row expansion.

Management of the job queue:

1. an idle worker requests access to the next pointer in the queue;
2. once given access, the worker takes the job and becomes busy;
3. the factor is added to the factors computed by the worker.

Dynamic load balancing works well in this way.

Random Boolean matrices of dimension 16 are generated. With 2 tasks, jobs are generated expanding the first two rows:

<table>
<thead>
<tr>
<th>#jobs</th>
<th>permanent</th>
<th>serial</th>
<th>2 tasks</th>
<th>speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>39</td>
<td>205676452</td>
<td>48</td>
<td>26</td>
<td>1.85</td>
</tr>
<tr>
<td>74</td>
<td>398844456</td>
<td>108</td>
<td>65</td>
<td>1.66</td>
</tr>
<tr>
<td>58</td>
<td>457676445</td>
<td>79</td>
<td>44</td>
<td>1.79</td>
</tr>
<tr>
<td>14</td>
<td>96908415</td>
<td>16</td>
<td>10</td>
<td>1.60</td>
</tr>
<tr>
<td>64</td>
<td>58417614</td>
<td>17</td>
<td>9</td>
<td>1.88</td>
</tr>
</tbody>
</table>

With 4 tasks, the first 3 rows are expanded, for a finer granularity:

<table>
<thead>
<tr>
<th>#jobs</th>
<th>permanent</th>
<th>serial</th>
<th>4 tasks</th>
<th>speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>278</td>
<td>282852334</td>
<td>45</td>
<td>24</td>
<td>1.88</td>
</tr>
<tr>
<td>420</td>
<td>268894344</td>
<td>95</td>
<td>52</td>
<td>1.83</td>
</tr>
<tr>
<td>521</td>
<td>39106098</td>
<td>14</td>
<td>7</td>
<td>2.00</td>
</tr>
<tr>
<td>321</td>
<td>77841276</td>
<td>37</td>
<td>20</td>
<td>1.85</td>
</tr>
<tr>
<td>359</td>
<td>1394427180</td>
<td>236</td>
<td>126</td>
<td>1.87</td>
</tr>
</tbody>
</table>
Random Boolean matrices of dimension 16 are generated. With 3 tasks, expanding the first 3 rows gives more jobs:

<table>
<thead>
<tr>
<th>#jobs</th>
<th>permanent</th>
<th>serial</th>
<th>3 tasks</th>
<th>speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>275</td>
<td>29320581</td>
<td>8</td>
<td>4</td>
<td>2.00</td>
</tr>
<tr>
<td>173</td>
<td>134237181</td>
<td>27</td>
<td>15</td>
<td>1.80</td>
</tr>
<tr>
<td>485</td>
<td>549654797</td>
<td>92</td>
<td>55</td>
<td>1.67</td>
</tr>
<tr>
<td>324</td>
<td>158044038</td>
<td>27</td>
<td>15</td>
<td>1.80</td>
</tr>
<tr>
<td>597</td>
<td>36928234</td>
<td>11</td>
<td>6</td>
<td>1.83</td>
</tr>
</tbody>
</table>

With 3 tasks, expanding only the first 2 rows gives fewer jobs:

<table>
<thead>
<tr>
<th>#jobs</th>
<th>permanent</th>
<th>serial</th>
<th>3 tasks</th>
<th>speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>111120492</td>
<td>15</td>
<td>8</td>
<td>1.88</td>
</tr>
<tr>
<td>38</td>
<td>116785084</td>
<td>44</td>
<td>22</td>
<td>2.00</td>
</tr>
<tr>
<td>39</td>
<td>224525956</td>
<td>35</td>
<td>18</td>
<td>1.94</td>
</tr>
<tr>
<td>53</td>
<td>67912248</td>
<td>9</td>
<td>5</td>
<td>1.80</td>
</tr>
<tr>
<td>66</td>
<td>497301012</td>
<td>112</td>
<td>56</td>
<td>2.00</td>
</tr>
</tbody>
</table>
44-core computer 2.2 GHz Intel Xeon E5-2699

On a random Boolean matrix of dimension 17, wall clock times are measured in seconds, jobs are generated expanding the first 3, 3, 4 rows:

<table>
<thead>
<tr>
<th>#jobs</th>
<th>permanent</th>
<th>#jobs</th>
<th>permanent</th>
<th>#jobs</th>
<th>permanent</th>
</tr>
</thead>
<tbody>
<tr>
<td>314</td>
<td>1413427296</td>
<td>188</td>
<td>412123207</td>
<td>1432</td>
<td>1452757932</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>#tasks</th>
<th>time</th>
<th>speedup</th>
<th>#tasks</th>
<th>time</th>
<th>speedup</th>
<th>#tasks</th>
<th>time</th>
<th>speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>284</td>
<td></td>
<td>1</td>
<td>152</td>
<td></td>
<td>1</td>
<td>431</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>172</td>
<td>1.65</td>
<td>2</td>
<td>86</td>
<td>1.76</td>
<td>2</td>
<td>238</td>
<td>1.81</td>
</tr>
<tr>
<td>4</td>
<td>89</td>
<td>3.19</td>
<td>4</td>
<td>45</td>
<td>3.78</td>
<td>4</td>
<td>122</td>
<td>3.53</td>
</tr>
<tr>
<td>8</td>
<td>49</td>
<td>5.80</td>
<td>8</td>
<td>24</td>
<td>6.33</td>
<td>8</td>
<td>63</td>
<td>6.84</td>
</tr>
<tr>
<td>16</td>
<td>25</td>
<td>11.36</td>
<td>16</td>
<td>13</td>
<td>11.69</td>
<td>16</td>
<td>33</td>
<td>13.06</td>
</tr>
<tr>
<td>32</td>
<td>15</td>
<td>18.93</td>
<td>32</td>
<td>8</td>
<td>19.00</td>
<td>32</td>
<td>19</td>
<td>22.68</td>
</tr>
<tr>
<td>64</td>
<td>11</td>
<td>25.81</td>
<td>64</td>
<td>6</td>
<td>25.33</td>
<td>64</td>
<td>14</td>
<td>30.79</td>
</tr>
</tbody>
</table>

Jan Verschelde (UIC)
Load Balancing by Work Stealing
FOSDEM 2018, 3 February

16 / 22
Problem Statement

- computing the permanent of a matrix
- high level parallel programming

Multitasking in Ada

- launching a crew of workers
- work stealing with multitasking
- application to polynomial system solving
work stealing with multitasking

The main data structure is a dequeue, with a beginning and an end. Although only one processor appends to the dequeue, both beginning and end must have semaphores.

- Underutilized processors pop from the beginning. Only one underutilized processor may pop, other underutilized processors must wait or skip to the following dequeue.
- New jobs are pushed to the end and popped from the end. If there is only one last job left, then the processor may have to wait for an underutilized processor.

We have a sequence of dequeues, one for every task.
Instead of generating jobs before the launching of the tasks, each task generates its own dequeue of jobs:

- If for an $n$-by-$n$ matrix, we consider only the first row, then task $k$ takes column $k$, for $k \leq n$.
- The permutations in the row expansions are serialized. For $p$ tasks, the $k$th task takes permutation $k$, $k + p$, $k + 2p$, etc. and computes the factors only for those permutations.
- Every task runs the same enumeration of permutations. The number of permutations in this enumeration is fixed in advance, before the launching of the tasks.

The load balancing in this static job assignment scheme then happens via work stealing. For $p$ processors, task $k$ starts looking at the dequeue of task $k + 1$, $k + 2$, etc. modulo $p$. 
Load Balancing by Work Stealing

1 Problem Statement
- computing the permanent of a matrix
- high level parallel programming

2 Multitasking in Ada
- launching a crew of workers
- work stealing with multitasking
- application to polynomial system solving
PHCpack is a package for Polynomial Homotopy Continuation. ACM Transactions on Mathematical Software achieved version 1.0 (Ada 83) as Algorithm 795, vol. 25, no. 2, pages 251–276, 1999.

**Blackbox solver:**

phc -b computes all isolated solutions of a polynomial system.

Version 2.0 was rewritten using concepts of Ada 95 and extended with arbitrary multiprecision arithmetic.

The current version is 2.3.48, last released December 2017.

Distributed under the GNU General Public License.

Public repository under version control at [https://github.com/janverschelde/PHCpack](https://github.com/janverschelde/PHCpack).

Code for this talk is in the folder src/Tasking.
applications to polynomial systems

- A permanent is a bound on the number of solutions.
  For example, the permanent gives the number of totally mixed Nash equilibria for any number of players with two pure strategies.

- We applied work stealing to intersections of polyhedral cones.
  This work is published in the Proceedings of PASCO 2017.