Problem Statement

Computer hardware has become parallel in order to run faster and more efficiently. Intel is developing a new parallel software and translator called CnC (Concurrent Collections) to make coding in parallel easier. With CnC, the user only specifies the work to be done and CnC automatically handles parallelization. **Our research goal was to see if CnC is efficient and usable when creating parallel code and converting serial code to parallel.**

How CnC Works

CnC uses a system of collections comprised of steps, items, and tags. In CnC, computational tasks runs on threads. In order to automatically determine which code can run parallel, CnC uses a graph system. This graph system allows the user to identify dependencies of code segments in the program. The parallelization of each of the segments (called tags) is handled automatically by CnC at runtime. In order to translate this abstract graph into concrete code, the graph is written in Intel’s proprietary textual notation. A CnC translator called cnc compiles this notation into a C++ header file.

Example CnC Code

```c
// The tag collection
<double avalue>>;

// Collections of output data
,double errors <double>>;
,int iterations <double>>;

// Step prescription
<avealue> :: (compute);

// Step execution
(compute) -> [errors];
(compute) -> [iterations];

// Input from the environment
env -> <avealue>;

// Output to the environment
[errors] -> env;
[iterations] -> env;
```

Example Main Code

```c
#include 'main.h'

int compute::execute(const double &
t, parallel_context & c) const{ 
    double error, a = t;
    int iter;

    //calls Poisson example function
    poisson(512, 1.0e-06, 99999,
    &error, &iter, a);
    return CnC::CNC_Success;
}

int main(int argc, char * argv[]){ 
    int j, M = atoi(argv[1]);
    double a;

    //creates a CnC context
    parallel_context c;

    //insert random a values into tags
    //run compute step on each tag
    for (j = 0; j < M; j ++){
        srand(j);
        a = 1000.0 * ((double)rand() / 
        (double)RAND_MAX);
        c.avalue.put(a);
    }
    c.wait();
    return 0;
}
```

Parameter Study

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a</td>
<td>iter</td>
<td>1 Thread</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>-----</td>
<td>-----------</td>
</tr>
<tr>
<td>135.44</td>
<td>594</td>
<td>1.72</td>
<td>2.97</td>
</tr>
<tr>
<td>274.75</td>
<td>477</td>
<td>1.38</td>
<td>2.65</td>
</tr>
<tr>
<td>486.90</td>
<td>369</td>
<td>1.09</td>
<td>2.35</td>
</tr>
<tr>
<td>561.38</td>
<td>367</td>
<td>1.08</td>
<td>2.26</td>
</tr>
<tr>
<td>700.98</td>
<td>330</td>
<td>0.97</td>
<td>2.20</td>
</tr>
<tr>
<td>840.19</td>
<td>300</td>
<td>0.89</td>
<td>2.02</td>
</tr>
<tr>
<td>840.19</td>
<td>300</td>
<td>0.89</td>
<td>2.02</td>
</tr>
<tr>
<td>916.46</td>
<td>287</td>
<td>0.86</td>
<td>1.95</td>
</tr>
<tr>
<td>Total</td>
<td>8.88</td>
<td>2.98</td>
<td></td>
</tr>
</tbody>
</table>

Times are in seconds. Extention of Poisson example [1] to $-\Delta u + a u = f$ with parameter $a > 0$. As $a$ increases, the conjugate gradient method requires fewer iterations and runtimes decrease.

Examples

<table>
<thead>
<tr>
<th>$M$</th>
<th>Worst Case</th>
<th>Best Case</th>
<th>Actual Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>2.97</td>
<td>2.303</td>
<td>2.98</td>
</tr>
<tr>
<td>64</td>
<td>31.75</td>
<td>20.333</td>
<td>21.19</td>
</tr>
<tr>
<td>256</td>
<td>137.82</td>
<td>86.835</td>
<td>88.44</td>
</tr>
<tr>
<td>1024</td>
<td>556.18</td>
<td>351.908</td>
<td>352.63</td>
</tr>
</tbody>
</table>

$M$ is the number of Poisson runs distributed across 8 threads. Worst case is $M$ times the maximum runtime. Best case is $M$ times the mean runtime.

Results

Conclusions

- CnC is a useful method for parallel programming
- Parallelism constraints are explicit. Parallelization is done automatically at runtime.
- Excels at parameter studies that vary in memory and runtime
- CnC’s graph concept is unique and easy to follow

References


Full technical report and code: HPCF–2011–14

www.umbc.edu/hpcf > Publications.


Acknowledgments

- Team members supported by a grant from the National Security Agency (NSA)
- REU Site [www.umbc.edu/hpcreu](http://www.umbc.edu/hpcreu)
- UMBC High Performance Computing Facility [www.umbc.edu/hpcf](http://www.umbc.edu/hpcf)