Parallel Programmability
and the **Cowichan Problems**

Andrew Borzenko
Cameron Gorrie

The Parallel Problem

- Parallel hardware is becoming ubiquitous faster than developers are learning how to use it effectively.
  - as a result, hundreds of parallel programming systems for various systems and programming languages have cropped up—with various degrees of polish, usefulness, and success
- Wilson [94] developed a set of "toy" problems to implement using various parallel systems to assess their degrees of programmability, dubbed the "**Cowichan problems**".
- We have implemented an updated version of the Cowichan problem set using two state-of-the-art parallel systems in order to assess ease of implementation and overall usefulness
- We define **programmability** a qualitative metric encompassing effort on the part of the application developer—how difficult it is to get ideas "onto paper" with the parallel system (intuitiveness).

Message Passing Interface

- Message Passing Interface (MPI) is a specification for an API that allows many computers to communicate with one another.
- Boost.MPI is a library for message passing in high-performance parallel applications.
- Boost MPI is not a completely new parallel programming library. Rather, it is a C++-friendly interface to the standard Message Passing Interface, the most popular library interface for high-performance, distributed computing.
- MPICH is an MPI implementation that efficiently supports various computation and communication platforms.
- There are other implementations of MPI, but we used MPICH since it is platform independent (in particular, it runs on windows)

Sources: boost.org, www.mcs.anl.gov

Threading Building Blocks

- TBB is a high-level parallelism library developed by Intel for C++ developers, incorporating many basic threading ideas not covered by system call interfaces
  - It also contains primitives and classes that implement a form of the data-parallel paradigm; refer to this as TBB/DP
- The **data-parallel** paradigm (DP) expresses work-sharing by doing exactly the same thing to (potentially) large collections of data
  - Worker sub-processes *cannot communicate*
  - Programmer defines the data domain (*where*), and the operation to do with that data (*what*)
  - **Key Insight:** TBB decides the *how* (i.e. splitting up the range onto different workers)


Best Scaling with 2 Threads/Processes and 4GB of Memory
Scaling Factor
2
1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
gauss
half
hull
invperc
life
mandel
norm
outer
product
randmat
sor
thresh
vecdiff
winnow
MPI
TBB

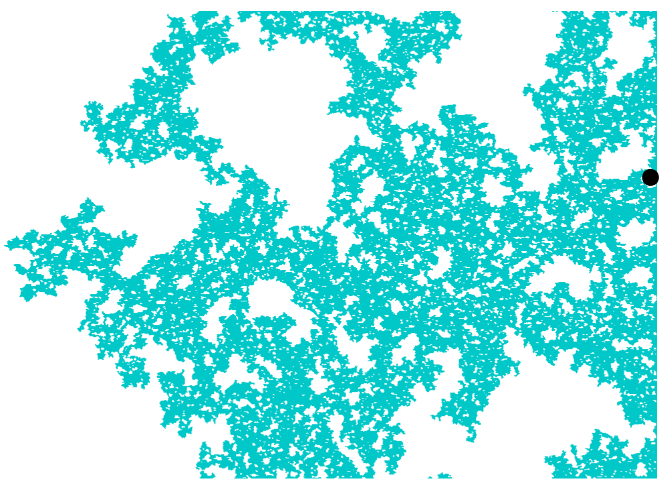
Performance Analysis

- Problems that are hard to parallelize in both: gauss, invperc, winnow
- Problems that perform better with MPI: mandel (due to use of task farm), thresh, product
- Problems that perform better with TBB: half, hull, life, norm, outer, randmat, sor, vecdiff
- Generally, **TBB performs better** because of the low cost of communicating the result—MPI communication is done between nodes that do not share any memory, while TBB uses threads and does use shared memory.

- Developers looking to effectively utilize parallel hardware aren't always starting at square one. The opposite is probably true, in fact—an existing serial codebase is being "parallelized".
- The natural question that arises from these developers is a simple one: "<u>can</u> my code be made parallel, or am I wasting my time?"

Some programming problems are **impossible** to effectively parallelize.

- The **Invasion Percolation** problem from the Cowichan problem can be solved in parallel, but the serial implementation is <u>orders of magnitude</u> faster. This is because every step of evaluation depends on the results of the previous step.



- Some programming problems are impossible to effectively parallelize, and —more broadly—some problems cannot be phrased in the "language" of a specific parallel system.

MPI: Practical Uses

- Running on supercomputers or **clusters**
- Algorithms should be restricted to those that have **very good** parallelization potential, due to slow communication speed
- Boost::MPI is good choice for algorithms that work on **enormous datasets** that cannot necessarily fit in the memory of a single node

---

- The best-case scenario for parallel speed-up was with the **matrix vector product** cowichan problem.
- We used 8 machines (on CDF), with 1 process per machine.
- The resulting speed-up was **6.4**, that is, the MPI implementation was able to solve the problem at 640% of the speed of the serial version.

MPI: Point-by-Point

ADVANTAGES

- Full support for both primitive and **user-defined** data types
- **Built-in** operations on arrays, vectors, strings
- Boost.Serialization facilitates **transfer of arbitrary data** between nodes
- Allows **heterogeneous operations** on aggregates
- Support for process groups
- **Synchronization is hidden from the user**
- Boost::MPI and MPICH are both very **well-maintained** and **platform independent**

DISADVANTAGES

- Very invasive code modification or restructuring required for existing applications, as **MPI must be used everywhere**
- **Slow startup/shutdown** when using a lot of nodes
- Performance **limited by communication** since memory is not shared between nodes
- Precisely the same executable <u>must be used</u> on every node.
- **Memory usage** can be a problem when running multiple nodes on the same computer
- Generally, requires **large datasets** to show performance benefits

TBB/DP: Point-by-Point

ADVANTAGES

- Simple set-up and **non-invasive code** modification needs for existing applications (very drop-in friendly)
- **Conceptually simple**—a low learning curve contributes to a good level of programmability
- Shared memory architecture is **familiar** to application developers
- **Well-maintained** by a large company, and **cross-platform**
- **Work is split automatically**; write for *n* processors rather than a specific #
- **Good parallel speed-up**, even for small datasets
- Implements **work-stealing**

DISADVANTAGES

- Expressing problems as data-parallel problems (identical sub-tasks) can be the most challenging part, and **some problems do not translate well** to this paradigm
- Work can only be split up locally; there is **no cluster or network capability**
- Work can only be split into new threads; therefore, as of now, there is **no support for GPU** programming

Code Examples

```
int lo, hi;     /* work controls */
int r, c;       /* loop indices */
int rank;

// work
if (get_block_rows_mpi (world, 0, nr, &lo, &hi))
{
  for (r = lo; r < hi; r ++) {
    result[r] = matrix[r * nc] * vector[0];
    for (c = 1; c < nc; c++) {
      result[r] += matrix[r*nc + c] * vector[c];
    }
  }
}

// broadcast result
for (rank = 0; rank < world.size (); rank++){
  if (get_block_rows_mpi
      (world, 0, nr, &lo, &hi, rank)) {
    broadcast (world, &result[lo], hi-lo, rank);
  }
}
```

MATRIX-VECTOR PRODUCT

Implementation of product using Boost::MPI

```
// use TBB to iterate over the given range
parallel_for(
    blocked_range2d<size_t,size_t>
    (0, BOARD_SIZE, 0, BOARD_SIZE),
    myGame, auto_partitioner());

...

void Game::operator()(const
blocked_range2d<size_t,size_t>& range)
const {

  // calculate this section of the game
  for (y in range.rows()) {
    for (x in range.cols()) {
      int peers = sumNeighbours(x, y);
      if (peers<2 || peers>3)
        (*second)[y][x] = 0;
      elif (peers == 3)
        (*second)[y][x] = 1;
      else
        (*second)[y][x] = (*first)[y][x];
    }
  }

}
```

GAME OF LIFE

Implementation of life using TBB/DP

Conclusions

- **MPI**, while being an extremely powerful platform, has a prohibitively large number of disadvantages that make it a poor choice for the wider audience of application developers.
- **TBB** and data parallelism is relatively simple to program, but lacks the ability to distribute work over a network or cluster. It is, however, a <u>good choice</u> if work will remain on the same computer.
- A novel library with a high degree of programmability that combines local and distributed parallel processing approaches would fill the gap.
  - an application could simply use both TBB and MPI for the same purpose—but this “solution” has a very <u>low</u> degree of programmmability due to its complexity and the extreme amount of effort required by the application developer.