Introduction to the WestGrid Development Environment

Jonatan Aronsson
WestGrid Site-Lead
University of Manitoba (Grex)

Getting Help

- Single Email Address: support@westgrid.ca
- Local Support Staff
- Online Documentation
  - www.westgrid.ca/support

Objective

- Provide you with:
  - An overview of development tools at WestGrid
  - Advice on how-to optimize code
  - Introduction to advanced optimization and parallelization
- Focus on the Intel Compiler Suite
- There is much more to talk about…

Outline

- Available compilers within WestGrid
- Optimization with compiler flags
- Strategies for advanced optimization
- Brief introduction to parallelization

Compilers at WestGrid

- All systems have the GNU and Intel compilers
- The PGI compilers are available on several WestGrid systems
- Use the compiler that works for your application
- Use multiple compilers if you develop your own code
  - Easier to find bugs
  - Make code more portable
- Compilers usually include debuggers and profilers

GNU Compilers

- The GNU compiler collection is free software and available for most operating systems and CPU architectures.
- C, C++, Fortran compilers are on all WestGrid systems.
  - Java, Ada, Go, and more are also available
- Support OpenMP and MPI parallelization

Intel Compiler Suite

- C, C++, Fortran
- Development Tools: Profiler, Memory checker, Debugger, Thread/MPI Analyzers
- Additional components (compatible with GNU)
  - Math Kernel Library (MKL)
  - Thread Building Blocks (TBB)
  - Integrated Performance Primitives (IPP)
- Parallelization support: MPI, OpenMP, Clik Plus

Ask the Compiler

- The intel compiler can provide an optimization report with the **-opt-report#** flag
  - # is a number between 1-3 (3 gives the most detailed information)
  - Report on how the code was optimized, for example
    - Which loops were unrolled or permuted
    - Deletion of unused code
  - Provide advice on beneficial code modifications

```
icc -opt-report1 -c test.c
```

Optimization Report

```
icc -opt-report1 -c test.c
```

Poor memory locality: Array of structures detected
Advice: Replacing Array of Structures by Structure of Arrays might help loopnest at lines: 160

Loop at line 144 completely unrolled by 10

test.c(190): loop was not vectorized: existence of vector dependence

Poor memory locality: Non-unit-stride memory reference detected
Advice: Data transposing might help loopnest at lines: 290

Loop at line 391 completely unrolled by 10 (pre-vector)

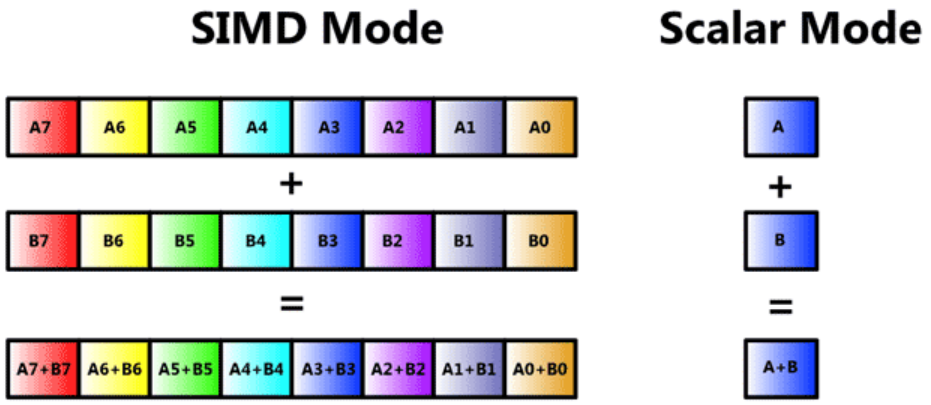
Loop at line 359 completely unrolled by 10

Basic Optimization

- By default, the compiler tries to minimize the compilation time while making debugging predicable.
- Optimization can be enabled with: -O1, -O2, -O3
  - Instructs the compiler to improve the performance (and/or code size)
  - Debugging may not be possible
  - -O1 enables basic optimizations
  - -O2 enables most optimizations
    - Recommended for general (non HPC) codes
  - -O3 enables -O2 and more aggressive strategies (e.g. loop transformations).
    - Recommend for heavy float-point calculations (HPC)

Typical Optimizations

- Perform auto-vectorization
  - Enables SIMD operations (SSE/AVX instructions on modern processors)
- Inlining of functions
- Loop transformations
  - Loop distribution
  - Loop interchange
  - Loop fusion
  - Loop unrolling
- Data pre-fetching



Source: software.intel.com

CPU Specific Optimizations

- Additional performance can be provided with CPU-specific optimizations
  - **-m<code>** Use all instructions up to <code> instruction set.
    - Optimize for both Intel and non-Intel processors
  - **-x<code>** Same as -m<code> but optimize for Intel processors only.
    - **-xsse4.2** Use up to the SSE4.2 instruction set (Intel Xeon 55xx or newer)
    - **-xhost** use all instructions on the host CPU
  - **-ax<code>** Add additional code paths for processors that have the <code> instruction set
    - One executable that is optimized for multiple CPU architectures

CPU Specific Optimizations

- The Bugaboo and Jasper clusters have mixed CPUs
  - Use: **-xsse3 -axsse4.2,sse4.1**
- On other WestGrid systems you can use **-xhost**

Interprocedural Optimization

- Interprocedural Optimization (IPO) analyzes the whole program to perform global optimizations, including
  - eliminate duplicated code
  - improve memory layout and locality.
- The Intel compiler support two modes
  - **-ip** Single-file IPO
  - **-ipo** Multi-file IPO (whole program optimization)
- Specify **-ipo** at both compilation and linking

```
icc -ipo -c test.c
icc -ipo test.o -o test
```

NOTE: -ipo can be very expensive for large codes and can fail. We recommend **-ip** in general.

IPO

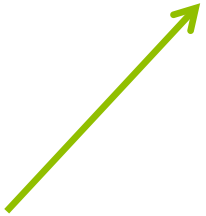
icc -ipo -opt-report1 -opt-report-phase=ipo

```
<mesh.c;22:25;IPO INLINING;cmpPeY;0>
INLINING REPORT: (cmpPeY) [2/14=14.3%]

<tree.c;-1:-1;IPO DEAD STATIC FUNCTION
ELIMINATION;ShiftInnerToInner;0>
DEAD STATIC FUNCTION ELIMINATION:
  (ShiftInnerToInner)
  Routine is dead static

<tree.c;28:39;IPO INLINING;buildTree;0>
INLINING REPORT: (buildTree) [11/14=78.6%]

  -> InitMesh(4) (isz = 69) (sz = 78 (21+57))
  -> initBiomials(13) (isz = 37) (sz = 40 (21+19))
```

-fast

- `-fast` is a general flag for optimizing for speed
  - Alias for:

```
-ipo -O3 -xhost  -no-prec-div  -static
```



Can improve floating-point divisions
by A/B = A*(1/B)

Static linking

Profile Guided Optimization (PGO)

- The compiler make general optimizations about your code without knowing how you use the code and/or what kind of input parameters you feed your program.
  - Those optimization may not be optimal and can even slow down your code
- PGO lets you provide a real program trace to the compiler in order to tell the compiler what to expect
- The profiled dataset should be statistically representative of the typical usage
  - Otherwise, it may harm the performance

PGO

- 3-phase procedure to compile with PGO
- 1) Compile with **-prof-gen** without optimizations
- 2) Run your code with a characteristic dataset
  - Ideally one that run quickly since the code is not optimized
  - This should generate a file **pgopti.dpi**
- 3) Re-compile with **-prof-use**

Example

gcc	29.9s
gcc -O3	21.5s
gcc -O3 -ftree-vectorize -march=native	20.4s
icc -O0	19s
icc -O3	13.2s
icc -O3 -xhost	12.6s
icc -O3 -xhost -ip	12.6s
icc -O3 -xhost -ipo	12s
ADD -prof-use	12s
ADD -fno-alias	7.7s
ADD -complex-limited-range	7.5s

Solves an N-body problem with 1,000,000 bodies using the Fast Multipole Algorithm

Dangerous optimizations

Advanced Optimization

- Beyond the automatic optimization, we will need to modify the source code
- Only minor changes may be required.
  - Embed hints (pragmas) to the compiler
  - No changes to the actual program code
- Strategy
  - Profile your code and identify hot spots
  - Focus your efforts on the hot spots (80/20 rule)
  - Use feedback from compiler (and profiler) to apply optimizations

Vectorization

```
for (i=0; i<n; i++)
    a[i] = b[i] * c[i];
```

The compiler does not know if **a, b, c** overlap in memory
=> No SIMD

```
#pragma simd
for (i=0; i<n; i++)
    a[i] = b[i] * c[i];
```

Tell the compiler that a, b, c are independent => SIMD

- Tell the compiler when vectorization is safe with **#pragma SIMD**
  - Many more options are available with the SIMD keyword
  - Fortran: **!DIR$ SIMD**
- Use **-vec-report#** (# = 1,…,6) to get advice

Array Notation

```
for (i=0; i<n; i++)
    a[i] = b[i] * c[i];
```

```
a[:] = b[:] * c[:];
```

- Another strategy for enabling vectorization is to use array notation
  - Provided with the Cilk Plus C/C++ extensions
    - **Fully supported in the Intel Compilers and in a branch of GCC 4.8**
  - Already in Fortran

More Optimization

- So far we have optimized our code by
  - Using compiler flags
  - Making code changes based on advice given by the compiler
  - Vectorizing code
- What's next?
  - Profile your code repeatedly
  - Optimize for memory/cache throughput (maximize FLOPs)
    - Often requires algorithmic redesign by a programmer who understands the hardware architecture! => Forget it unless you are black belt
  - Use existing libraries with pre-optimized routines
  - Parallelize code

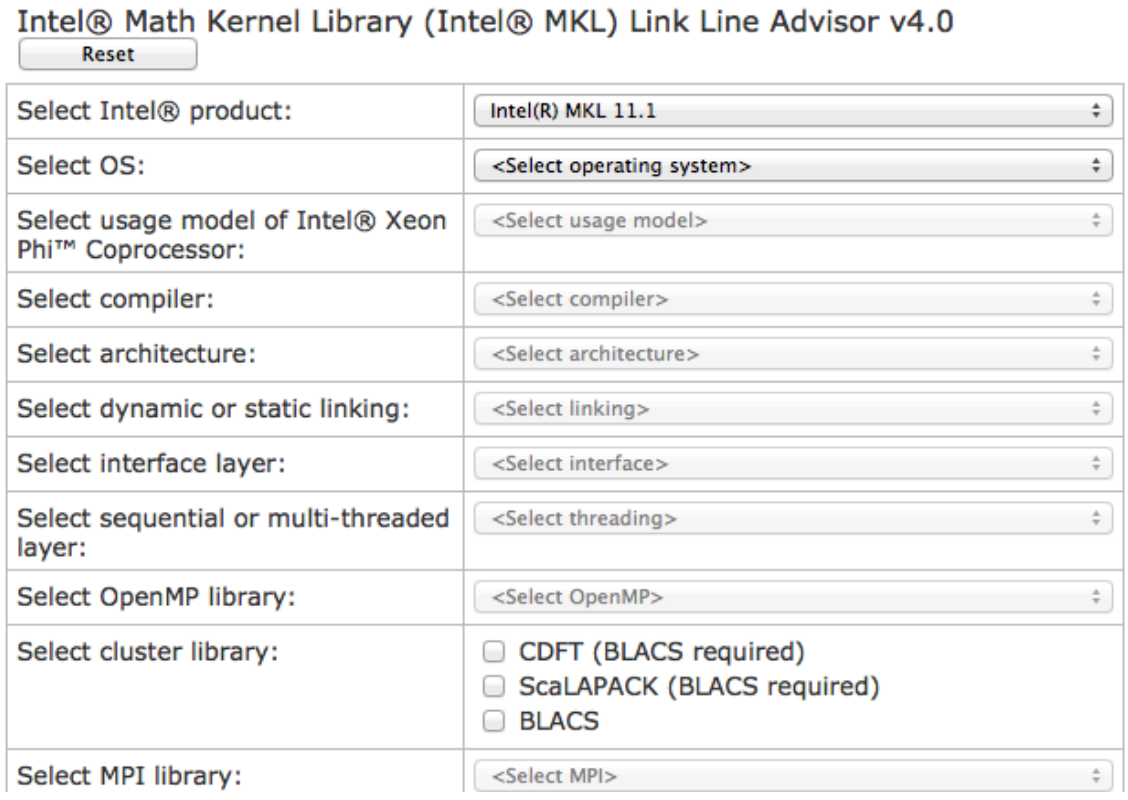
Intel MKL

- The Intel Math Kernel Library (MKL) library is available on all WestGrid systems and include:

- Linear Algebra
  - BLAS
  - LAPACK
  - ScaLAPACK
  - Sparse solvers
- FFT
  - Multidimensional
  - FFTW interface
  - Cluster FFT
- Vector math
- Random Number Generators
- Statistical Methods
- Others
  - Nonlinear optimization
  - PDE Solvers
  - Data fitting
  - More …

Using MKL

Use: http://software.intel.com/en-us/articles/intel-mkl-link-line-advisor



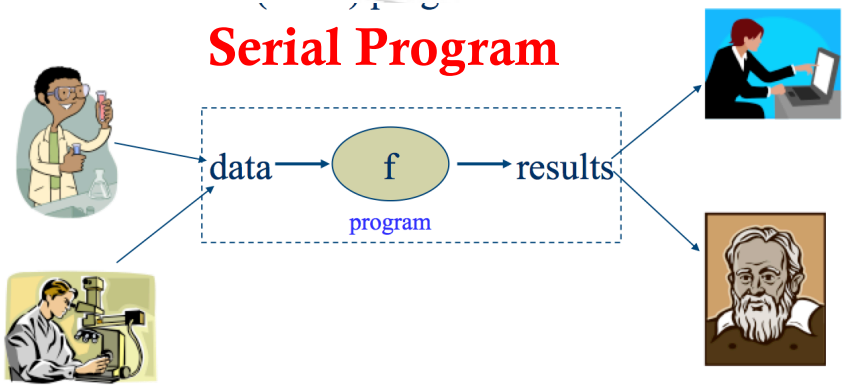
More Libraries

- WestGrid supports a large number of libraries
  - See: www.westgrid.ca/support/software
  - We install libraries when users request them
- Look for a library that can do the job before developing new code
  - Libraries are often well optimized
  - Save time

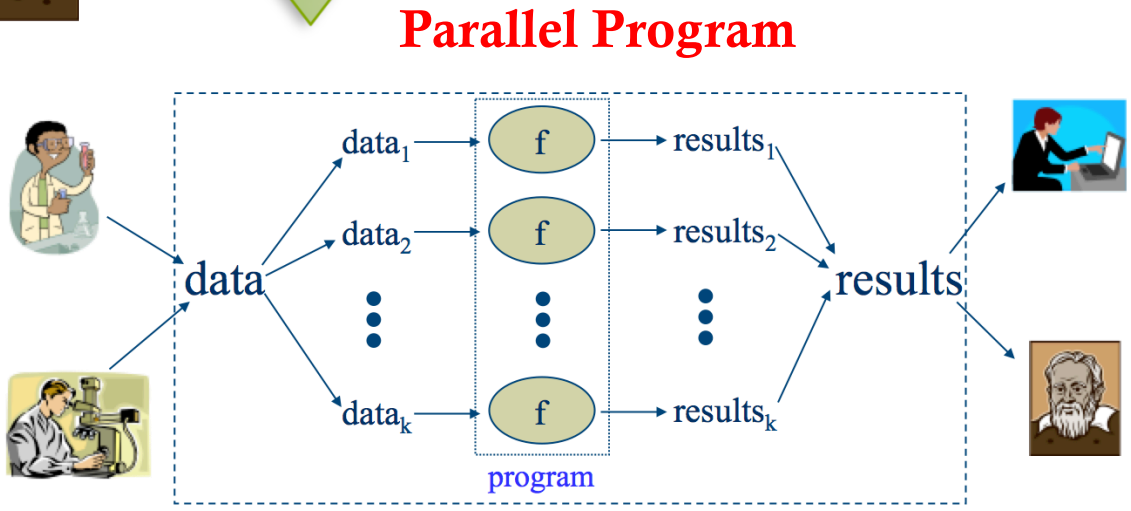
Parallelization

- To further improve the performance of our code, we may write parallel code
- The challenge in parallel programming is to take our algorithm and divide the computations into pieces that can run across multiple processors
  - The most common approach is to divide up the data
  - Another approach is divide up the algorithm
- Two main programming models
  - Shared memory
  - Distributed memory

Common Scenario

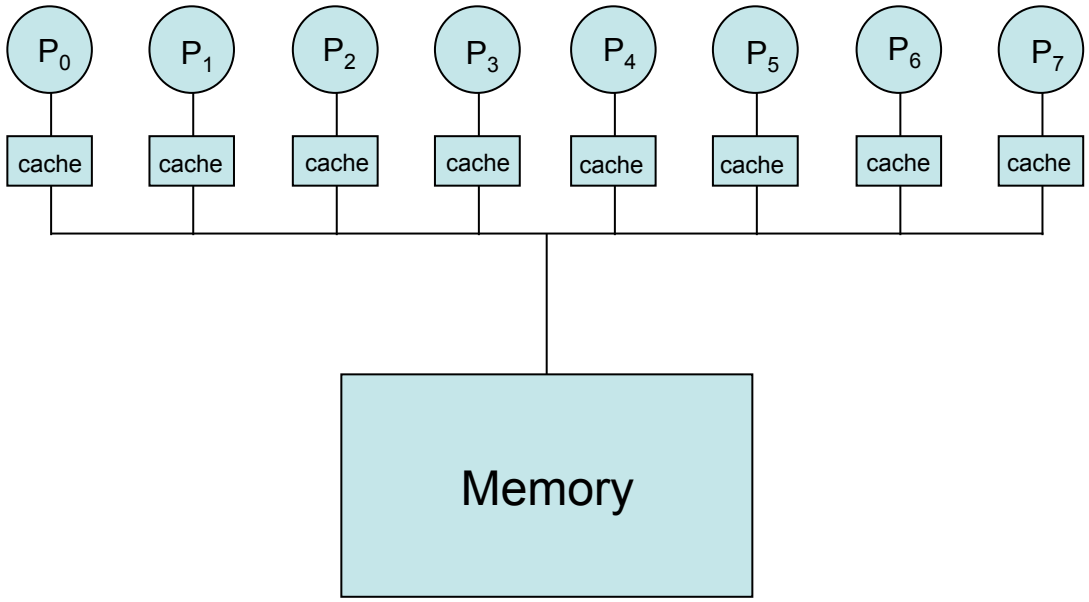


Parallel Programming



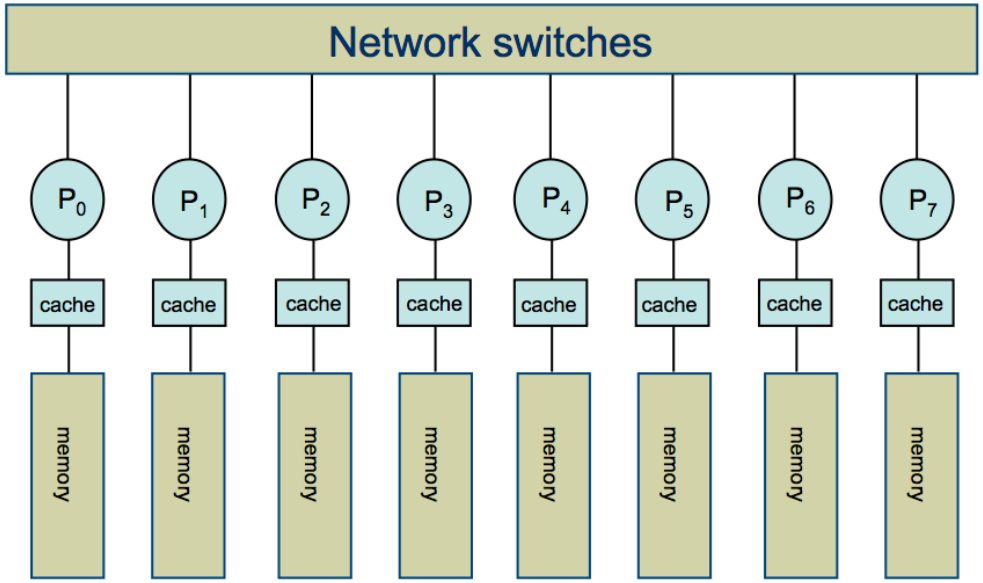
Figures courtesy of Peter Graham

Shared Memory Model



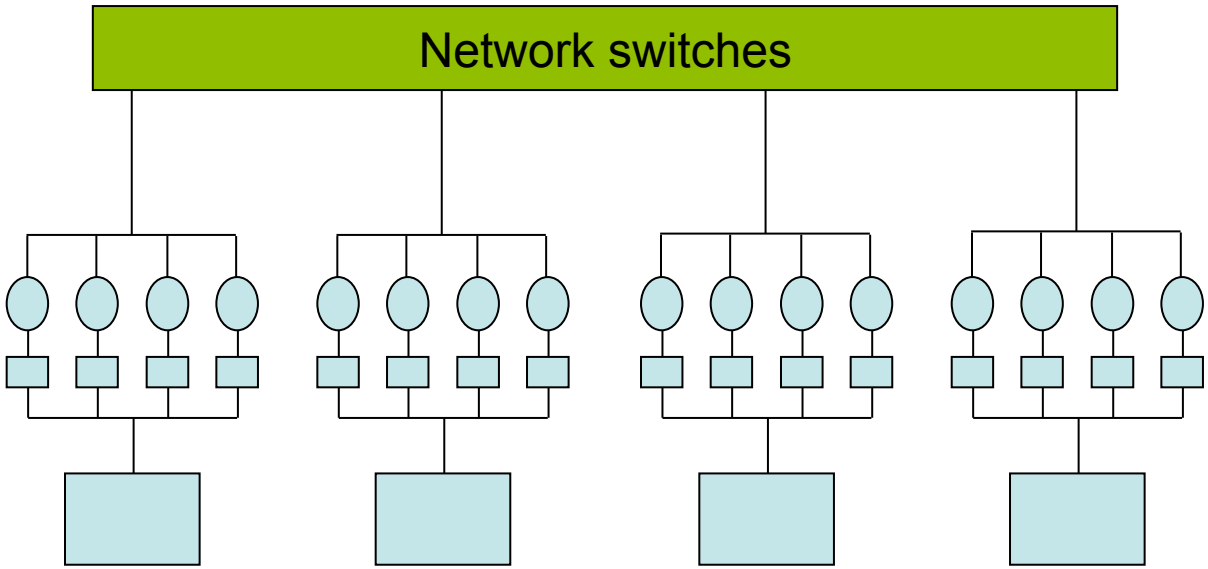
- Scale up to a single node
  - E.g. 12 cores on Grex or 2048 cores on Hungabee
- Familiar environment
- Limited scalability
- Programming tools
  - Automatic parallelization
  - Pthreads
  - OpenMP
  - Clik Plus

Distributed Memory Model



- Scalable and inexpensive
  - Can run from 1 to 1000's of cores within WestGrid
- Limited memory per core
- Often harder to program
- Use MPI (Message Passing Interface) to program

Hybrid Model



- Becoming more important
  - E.g. many-core, accelerators
- More programming effort
- Flexible
  - Best of both?

Summary

- Strategies for optimizing code:
  - Use compiler flags
    - Test and see what works for your code
  - Ask the compiler and/or use a profiler
    - Optimize hot spots (iteratively)
  - Use existing libraries for common routines
  - Parallelize
- Documentation
  - Use man pages for the compilers: **`man icc, man ifort, man gcc`**
  - www.westgrid.ca/support
  - Ask WestGrid Support

Getting Support

- Single Email Address: support@westgrid.ca
- Local Support Staff
- Online Documentation
  - westgrid.ca/support

Thank you for your attention! Questions?