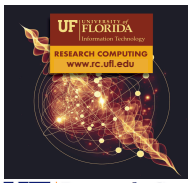
UF Research Computing: Overview Programming tools



Ying Zhang
yingz@ufl.edu

4/2/2015

UF Research Computing
Information Technology
Home of High-Performance Computing and HiPerGator

What Is Computer Program

- A sequence of instructions
  - Code
- Written in a computer language
  - C, Fortran, Java, or Python
- Perform specified tasks
  - Algorithm

Programming Environment

- Software setup for writing, compiling and executing programs
  - Compilers
    - C, C++, Fortran, etc
  - Interpreter
    - Matlab, Perl, Python, PHP, etc.
  - Other tools
    - Debuggers
    - Profilers
    - Performance analyzers

Serial VS Parallel

- Serial Computing
  - Instructions are executed sequentially
  - On a single processor
- Parallel Computing
  - A problem is broken into independent components and can be solved concurrently
  - Each Components execute simultaneously on different processor or threads

Programming Tools at RC

- Compilers
- Debuggers
- Performance profilers

Compilers

- Compilers available
  - Intel Compiler Suite
    - Online documentation
      https://software.intel.com/en-us/articles/new-user-compiler-basic-usage
    - Modules
      - `module spider intel`
  - GNU Compiler Collection
    - Online documentation
      https://gcc.gnu.org/onlinedocs/gcc-4.7.4/gcc/
      https://gcc.gnu.org/fortran/
    - Modules
      - `Module spider gcc`

Intel Compiler Suite

- For serial programs
  - `module load intel/2013`
- Usage
  - Compile C code: icc
    - e.g.: `icc -O1 mytest.c -o mytest`
  - Compile C++ code: icpc
    - e.g.: `icpc -g mytest.cpp -o mytest`
  - Compile Fortran code: ifort
    - e.g.: `ifort -O2 mytest.f90 -o mytest`

Intel Compiler Suite

- For MPI parallel programs
  - Modules
    - `module load intel/2013 openmpi/1.6.5`
  - Usage
    - Compile C code: mpicc
      - e.g.: `mpicc -O1 mytest.c -o mytest`
    - Compile C++ code: mpicxx
      - e.g.: `mpicxx -g mytest.cpp -o mytest`
    - Compile Fortran code: mpif77, mpif90
      - e.g.: `mpif90 -O2 mytest.f90 -o mytest`
- For OpenMP programs
  - `module load intel/2013`
  - `icc/icpc/ifort -openmp mytest.c/.cpp/.f90 -o mytest`

Intel Compiler Suite

- Basic compiler options
  - -g: debugging
  - -O#: optimization
    - -O0: no optimization
    - -O1: optimize for speed without increase code size
    - -O2: default, optimize for speed, eg. vectorization
    - -O3: high-level optimizer, prefetch, loop unrolling
  - Support for *target:* SSE2, SSE3, SSE4.1, SSE4.2, AVX
    - *-xtarget*: for Intel architecture only
    - *-mtarget*: for any architecture supporting *target*, recommended on HiPerGator

***SSE**: Streaming SIMD Extensions; **AVX**: Advanced Vector Extension*

Intel Compiler Suite

Compiler Based Vectorization

Feature	Extension
Intel® Streaming SIMD Extensions 2 (Intel® SSE2) as available in initial Pentium® 4 or compatible non-Intel processors	sse2
Intel® Streaming SIMD Extensions 3 (Intel® SSE3) as available in Pentium® 4 or compatible non-Intel processors	sse3
Supplemental Streaming SIMD Extensions 3 (SSSE3) as available in Intel® Core™2 Duo processors	ssse3
Intel® SSE4.1 as first introduced in Intel® 45nm Hi-K next generation Intel Core™ micro-architecture	sse4.1
Intel® SSE4.2 Accelerated String and Text Processing instructions supported first by by Intel® Core™ i7 processors	sse4.2
Extensions offered by Intel® ATOM™ processor : Intel SSSE3 (1!) and MOVBE instruction	sse3_atom
Intel® Advanced Vector Extensions (Intel® AVX) as available in 2nd generation Intel Core processor family – code name Sandy Bridge	AVX
Intel® Advanced Vector Extensions (Intel® AVX) as code-name Ivy Bridge and in code-name Haswell (available only in compilers v13+, fall 2012)	CORE-AVX-I CORE-AVX2

Software & Services Group
Developer Products Division

GNU Compiler Collection (GCC)

- For serial programs
  - Module
    - `module load gcc/4.7.2`
  - Usage
    - Compile C code: gcc
      - e.g.: `gcc -O1 mytest.c -o mytest`
    - Compile C++ code: g++
      - e.g.: `g++ -g mytest.cpp -o mytest`
    - Compile Fortran code: gfortran
      - e.g.: `gfortran -O2 mytest.f90 -o mytest`

GNU Compiler Collection

- For MPI parallel programs
  - Modules
    - `module load gcc/4.7.2 openmpi/1.6.5`
  - Usage
    - Compile C code: mpicc
      - e.g.: `mpicc -O1 mytest.c -o mytest`
    - Compile C++ code: mpicxx
      - e.g.: `mpicxx -g mytest.cpp -o mytest`
    - Compile Fortran code: mpif77, mpif90
      - e.g.: `mpif90 -O2 mytest.f90 -o mytest`
- For OpenMP programs
  - Module
    - `module load intel/2013`
  - Usage
    - `gcc/g++/gfortran -fopenmp mytest.c/.cpp/.f90 -o mytest`

GNU Compiler Collection

- Basic compiler options
  - -g: debugging
  - -O#: optimization
    - -O0: default, reduce compilation time
    - -O1: reduce code size and execution time
    - -O2: default, optimize for speed
    - -O3: O2 + prefetching, loop unrolling, vectorization, etc.
    - -Ofast: O3 + ffast-math and other Fortran related optimization
  - Support for: SSE2, SSE3, SSE4.1, SSE4.2, AVX
    - -mtune=*cpu-type*: generic, native, corei7, core-avx-I …
    - -msse, -msse2, -msse3, -msse4.1, -mavx …

Debuggers

- Debugging
  - Locate and remove errors or abnormalities
- Debugger available on HiPerGator
  - Allinea DDT
  - Intel Inspector XE
  - Valgrind

Allinea DDT

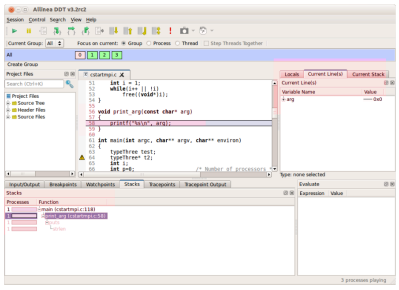
- A graphical distributed debugging tool
  - Serial code
  - Multithreaded code
  - Multiprocess code
  - C, C++, Fortran, CUDA
- Module required
  - `module load ddt`
  - Latest version: 5.0

Allinea DDT

- Flow control
  - Control program progress
  - Static analysis at thread and process level
  - Identify and fix problems
- Data monitoring
  - Track variables
  - Detect memory errors
  - Check data calculation
  - Trace points

Allinea DDT

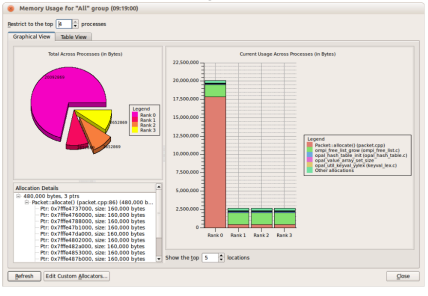
Flow control and data monitoring



Source: https://computing.llnl.gov/tutorials/allineaDDT/Examples.pdf

Allinea DDT

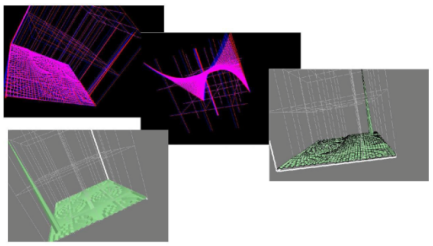
Memory usage analysis



Source: https://computing.llnl.gov/tutorials/allineaDDT/Examples.pdf

Allinea DDT

Multi-Dimensional Array Viewer



Source: https://www.allinea.com/user-guide/forge/ViewingVariablesAndData.html

Valgrind

- A memory mismanagement detector
- Detect
  - Memory leak
  - Deallocation errors
  - Use uninitialized memory
  - Read/Write freed memory
  - Mismatched allocation/deallocation and syntax
    - `malloc/new/new[] vs free/delete/delete[]`
- Cons
  - No bound checking for static arrays
  - Execution specific

Valgrind

- Required modules
  - `module spider valgrind`
  - For serial programs
    - `module load gcc/4.7.2 valgrind`
  - For parallel programs
    - `module load intel/2013 openmpi/1.6.5 valgrind`
    - `module load gcc/4.7.2 openmpi/1.6.5 valgrind`
- Usage
  - For serial programs,
    - e.g.: `valgrind --leak-check=yes myprog input`
  - For parallel programs
    - e.g.: `mpirun -np 4 valgrind --leak-check=yes myprog input`

Valgrind

- Example

```
#include <stdlib.h>

void f(void)
{
   int* x = malloc(10 * sizeof(int));
   x[10] = 0;        // problem 1: heap block overrun
}                    // problem 2: memory leak -- x not freed

int main(void)
{
   f();
   return 0;
}
```

- Valgrind output

```
==19182== Invalid write of size 4
==19182==    at 0x804838F: f (example.c:6)
==19182==    by 0x80483AB: main (example.c:11)
==19182==  Address 0x1BA45050 is 0 bytes after a block of size 40 alloc'd
==19182==    at 0x1B8FF5CD: malloc (vg_replace_malloc.c:130)
==19182==    by 0x8048385: f (example.c:5)
==19182==    by 0x80483AB: main (example.c:11)
```

Source: http://valgrind.org/docs/manual/quick-start.html#quick-start.prepare

Performance Profiling and Evaluation

- Dynamic program analysis that reveals features in the program and interactions between software and hardware
  - The frequency and duration of function calls & loops
  - Calculation speed
  - Memory usage
  - Communication pattern
  - Multithreading analysis
  - Load balance and efficiency
- Goal:
  - Code optimization

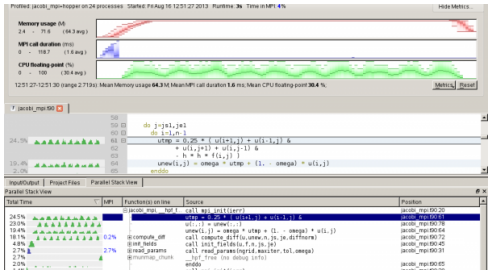
Performance Profiling & Evaluation Tools

- Allinea MAP
  - `module load ddt`
- TAU
  - `module load intel/2013 tau`
  - `module load intel/2013 openmpi/1.6.5 tau`
- Intel Toolkit
  - VTune Amplifier XE
  - Advisor XE

Allinea MAP

- A graphical distributed profiler
  - Detect bottlenecks at source level
  - Support
    - Serial, multithreaded, and multiprocess code
    - C, C++, Fortran 90
  - Performance metrics
    - Memory usage
    - Floating Point calculations
    - MPI call performance
  - Easy to use, no instrumentation or recompilation

Allinea MAP



Source: https://www.nersc.gov/users/software/debugging-and-profiling/MAP/

TAU

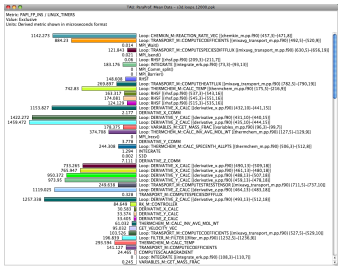
- Tuning and Analysis Utilities
- Integrated Performance Toolkit
  - Performance profiling and tracing
    - Both serial and parallel programs
  - Source code instrumentation
    - Adds probes to collect performance data
    - Manual or automatic
  - Comprehensive data analysis and management
    - Include hardware performance counters via PAPI
  - Performance data visualization
    - Paraprof
  - Open source
    - http://www.cs.uoregon.edu/research/tau/home.php

TAU

- Performance analysis
  - Time consumption – identify bottlenecks
    - At source level: loops, functions calls
    - I/O, communication, computation
  - Memory usage
    - Allocation/deallocation, cache misses
  - Floating point operations
    - FLOP rate
  - Cross process comparisons
    - Load balancing and scaling

TAU

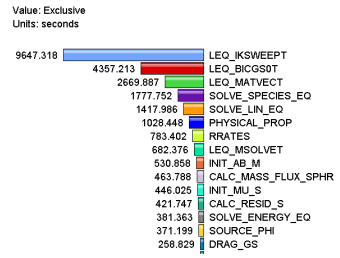
- Identify potential bottlenecks



Source: Sameer Shende, TAU Performance System Tutorial

TAU

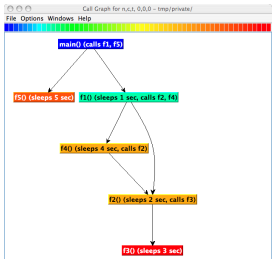
- How much time is spent in each routine?



Source: Sameer Shende, TAU Performance System Tutorial

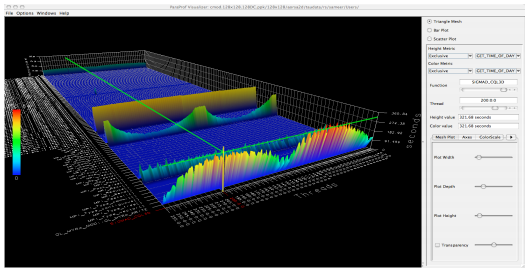
TAU

- Program call graph



Source: Sameer Shende, TAU Performance System Tutorial

TAU ParaProf 3D Profile Browser



Source: Sameer Shende, TAU Performance System Tutorial

Intel Parallel Studio

Phase	Productivity Tool	Feature	Benefit
Advanced Parallel Design	**Intel® Advisor XE**	Analyze existing code base and find opportunities for parallelization.	Easier analysis and performance heuristics, find compute hotspots and check for parallelization strategies.
Advanced Build and Debug	**Intel® Composer XE**	C/C++ and Fortran compilers, performance libraries, and parallel models	Application performance, scalability and quality for current multicore and future many-core systems.
Advanced Verify	**Intel® Inspector XE**	Memory & threading error checking tool for higher code reliability & quality	Increases productivity and lowers cost, by catching memory and threading defects early
Advanced Tune	**Intel® VTune™ Amplifier XE**	Performance Profiler to optimize performance and scalability	Removes guesswork, saves time, makes it easier to find performance and scalability bottlenecks Combines ease of use with deeper insights.

Source: Intel Parallel Studio Webinar

Intel Inspector XE

- Graphical software "Correctness Analyzer"
- Memory Error
  - Memory leaks
  - corruptions
  - allocation/deallocation
  - Inconsistent memory usage,
  - Illegal memory access
  - Uninitialized memory read
- Threading error
  - Data races
  - Deadlocks

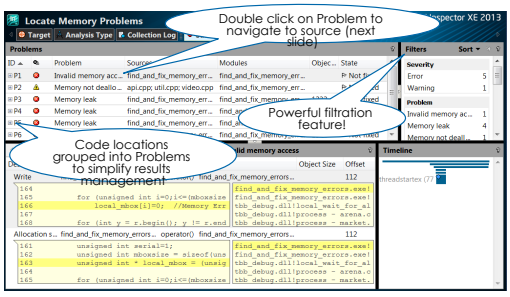
Intel Inspector XE

- Key features

Feature	Details
Analyses	• Dynamic Memory Analysis • Dynamic Threading Analysis
GUI	• Microsoft Visual Studio IDE integration (2008, 2010, or 2012) • Stand alone GUI on both Windows and Linux
Compilers supported	• Microsoft* Visual* C++ .NET* • Intel® Parallel Composer and Intel® Composer XE • Gcc
OS	• Windows XP, Vista, 7, 8 • Linux (various distros)
Languages	• C/C++ • C# • Fortran

Source: Intel Parallel Studio Webinar

Intel Inspector XE

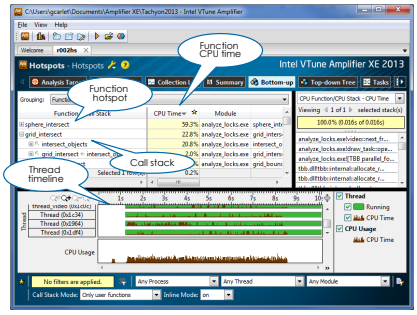


Source: Intel Parallel Studio Webinar

Intel VTune Amplifier XE

- Performance analyzer
  - Bottleneck identification
  - Source level performance data
  - Thread profiling
  - Hardware event based sampling
  - Support
    - C/C++, Fortran, Java, .NET
    - GCC, Intel Compiler, Windows
    - Serial and parallel programs
    - Linux and Windows

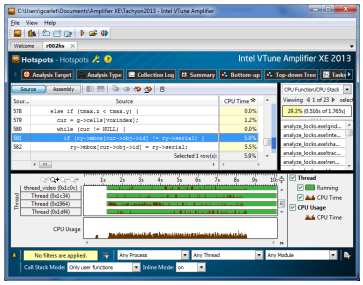
Intel VTune Amplifier XE



Source: Intel Parallel Studio Webinar

Intel VTune Amplifier XE

Source view – identify hotspots



Source: Intel Parallel Studio Webinar

Intel Advisor XE

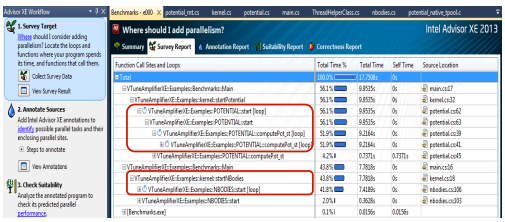
- Threading design and prototyping
  - Analyze, design, tune the threading design in multitheaded program
  - Identify the code location for parallelism
  - Identify synchronization errors
  - Predict threading errors and scaling
  - Separate design and implementation
- Support
  - C, C++, Fortran, C#
  - Linux, Windows

Intel Advisor XE

- Typical workflow
  - Survey
  - Add annotations
  - Model performance suitability
  - Check correctness
  - Add parallel framework

Intel Advisor XE

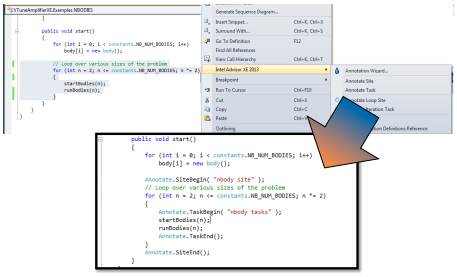
1. Survey



Source: Intel Parallel Studio Webinar

Intel Advisor XE

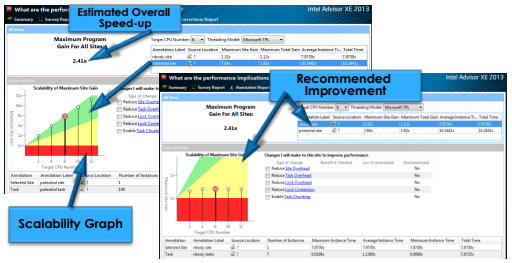
2. Add Annotation



Source: Intel Parallel Studio Webinar

Intel Advisor XE

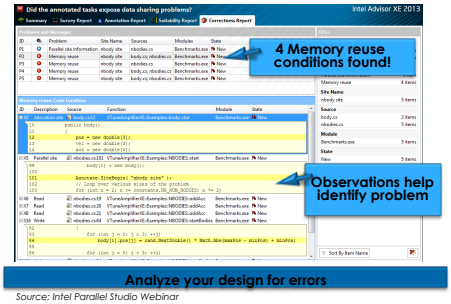
3. Add Annotation



Source: Intel Parallel Studio Webinar

Intel Advisor XE

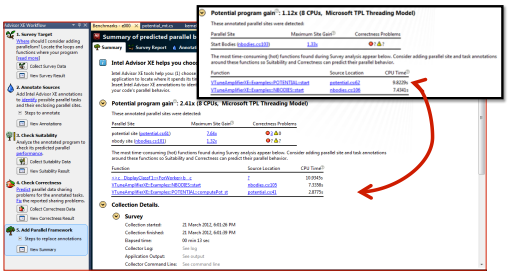
4. Check correctness



Source: Intel Parallel Studio Webinar

Intel Advisor XE

5. Add parallel framework



Source: Intel Parallel Studio Webinar

UF Research Computing

- Help and Support
  - https://my.it.ufl.edu
    - For any kind of question or help requests
  - http://wiki.rc.ufl.edu
    - Documents on hardware and software resources
    - Various user guides
    - Many sample submission scripts
  - http://rc.ufl.edu
    - Frequently Asked Questions
    - Account set up and maintenance