Module 11: The “lastprivate” Clause

Lecture 22: Intel Compilers and Threading Tools

The Lecture Contains:

Previous Next

Intel Compilers and Threading Tools

Motivation

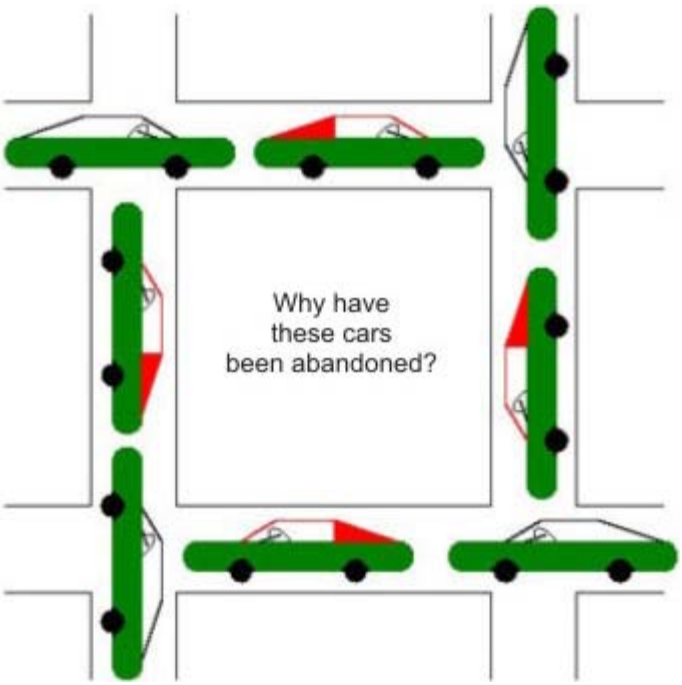
- The future is Multicores.

- Programmers must inevitably write parallel program constructs.
  - Complexity is way higher than sequential programming.
- What about existing sequential programs?
  - How to parallelize them?

Previous Next

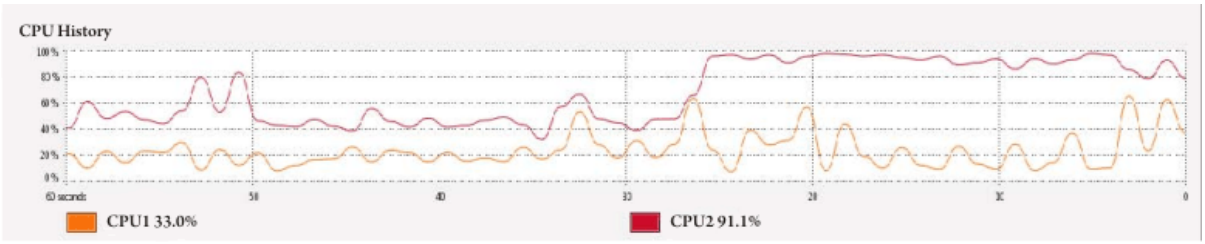
Some Issues Involved in Parallel Programming

- Deadlocks, Livelocks.



Contd . . .

- Data Races (Synchronization)
  - A race condition is a kind of error condition peculiar to parallel programs. The outcome of a program changes as the relative scheduling of units of execution (UEs) varies. This is often the result of modications to a shared or global variable by more than one UE without an appropriate synchronization mechanism (such as a barrier, mutex, or semaphore).
- Load Balancing



Previous Next

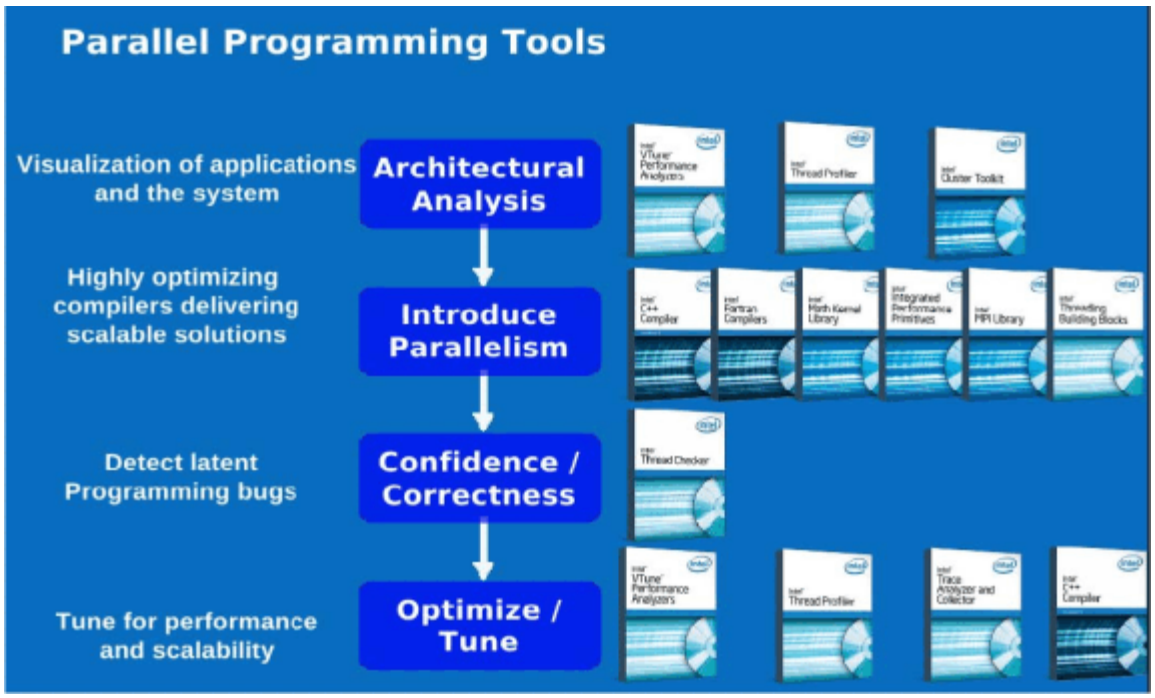
Contd . . .

- Core Utilization
  - The application must try to utilize 100% of the units of execution available.
- Correctness. (Consistency models)
  - How do we decide if the output is correct or not.
  - Must follow some rules.
  - Sequential Consistency, Linearizability
- Lock Free and Wait Free
  - A method is said to be wait-free if it guarantees that every call nishes its execution in a nite number of steps.
  - A method is said to be lock-free if it guarantees that innitely often some call nishes in a nite number of steps.

We Need Help

- Tools can help us nding out these issues and in some cases, resolve them too.
- Intel® Corporation has developed some tools for the development of multi-threaded applications under their Higher Performance Computing Category.

Intel Tools



Build Correct Optimize Cluster

- Build
  - Intel Compilers
- Correct
  - Intel Thread Checker
- Optimize
  - Intel Vtune Performance Analyzer
  - Intel Thread Proler
- Cluster
  - Intel Cluster Toolki

Previous Next

Module 11: The “lastprivate” Clause

Lecture 22: Intel Compilers and Threading Tools

Intel Compiler

- Based on The Programming Language
  - Intel C++ Compiler
  - Intel Fortran Compiler

- Features
  - Multi-Threaded Application Support (OpenMP)
  - High Performance Parallel Optimizer (HPO)
  - Automatic Vectorization
  - Prole-Guided Optimization (PGO)
  - Interprocedural Optimization (IPO)

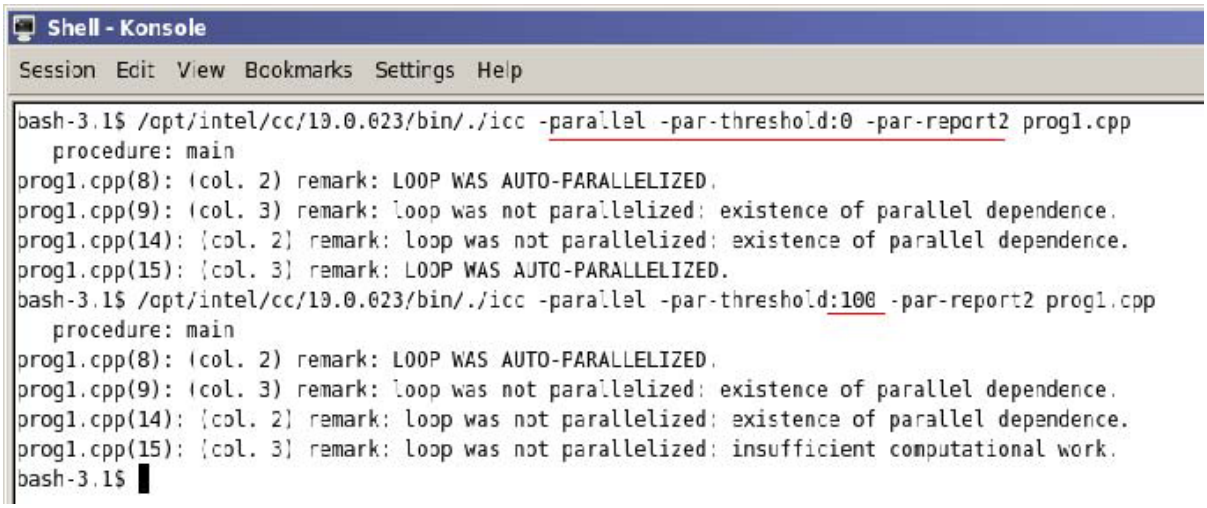
Example Code

prog1.cpp

```
1  using  namespace  std;
2
3  int  main() {
4
5          int  x[ 100000] ;
6          int  a[ 100000] , b[100000 ], c [100000]m;
7
8          for ( int i-0; i<100000; i+- {
9                  for ( int j-0; j<100000; j++ {
10                         x [ i ] = i ;
11                 }
12         }
13
14         for ( int i =0;  i< 100000; i++) {
15                 for ( int j =0;  j< 1000; j++) {
16                         a [ j ] =b [ j ] + c [ j ];
17                 }
18         }
19 }
```

Previous Next

Intel C++ Compiler Output View



Intel C++ Compiler Tips

- The default compilation is with O2 (level-2 optimization) flag set.
- The compiler autoparallelizes only if O2/O3 flag is set.
- Remember to set -par-report2/3 flag while autoparallelization for a complete diagnosis report.
- It is better to use O0 flag for beginners of OpenMP because some optimizations(code motion) can produce unexpected results.
- Refer to man pages when in doubt.

Previous Next

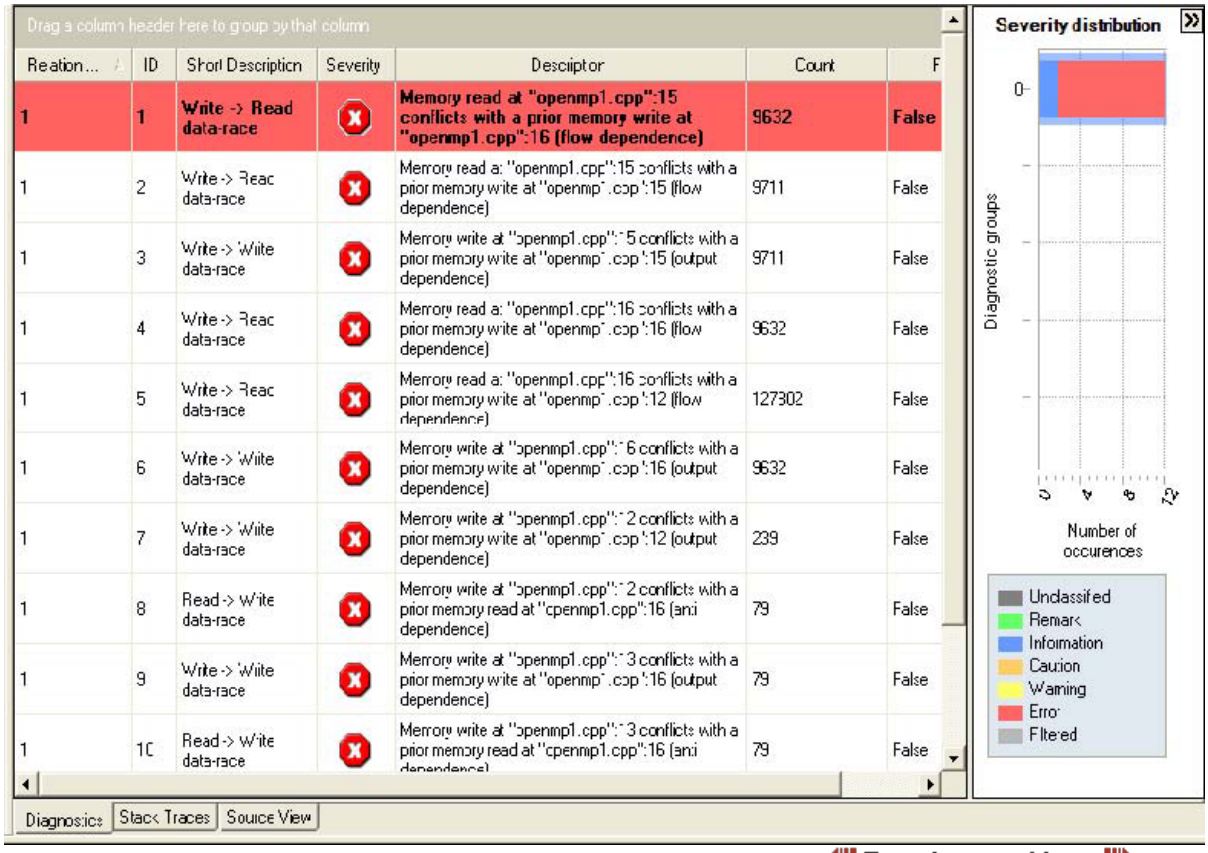
Module 11: The “lastprivate” Clause

Lecture 22: Intel Compilers and Threading Tools

Intel Thread Checker

- Features
  - Intel Thread Checker detects data races, deadlocks, stalls, and other threading issues. It can detect the potential for these errors even if the error does not occur during an analysis session.
  - Pinpoint the function, context, line, variable, and call stack in the source code to aid analysis and repair of bugs.
  - Identify nearly impossible-to-nd data races and deadlocks using an advanced error detection engine. Helps to reduce untraceable errors.
  - Errors do not need to actually occur to be detected. Make the code as more robust.

Intel Thread Checker Output View



Previous Next

Example Code

```
#include <pthread.h>
int Count;

void *findCount()
{
        Count++; //Flow, Anti, Output Dependency
}
int main()
{
    int i, rc;
    pthread_t h[2];

    Count = 0;

    for ( i = 0; i < 2; ++i)
    {
        rc = pthread_create (&h[i], 0, findCount, NULL);
    }

    for ( i = 0; i < 2; ++i)
    {
        rc = pthread_join (h[i], 0);
    }
    return 0;
}
```

Previous Next