Curriculum Development for Multi-core Software Technology at Peking University

Prof. Zhonghai Wu
School of Software and Microelectronics
Peking University
http://www.ss.pku.edu.cn/multicore
August 2008
Cooperation in Field of Multi-core Technology Between Intel and SSPKU

- SSPKU and INTEL start multi-core trainer training class in 2004 for more than 30 teachers from Chinese well-known universities.
- SSPKU set up PKU-INTEL Multi-core Lab in 2006, and introduce multi-core technology into several courses such as Computer Architecture, Internet Programming, and Parallel Computing.
- SSPKU got equipment donation and grant funding for multi-core curriculum development in 2006.
- SSPKU sent 3 multi-core teachers to INTEL China as visiting Scholars. One of them joined 2007’s International Workshop on Intel Multi-core Technology in Unite States.
- SSPKU send more than ten graduate students of PKU-INTEL joint lab to do their multi-intern in Intel China each year.
- SSPKU introduced a new course in 2006, “Software Development based on Intel Multi-core Architecture” for senior undergraduate and graduate students.
- SSPKU joined in the writing and editing of textbook “Multi-core Programming” in 2007.
Objectives of the Course

• Multi-core technology is involved in several courses which include Computer Architecture, Operating System, Compiling Technology, Software Development Methodology, Software Design Schema, Programming Model, and Parallel Computing. It is difficult for software engineers to be skillful on the multi-core based software development in a short time without systemic training.

• To Training graduate students and advanced software engineers who are well up in multi-core technology and absorbed in software development on multi-core platform systematically, we introduce a new course, “Multi-core Software Development Technology” in our school.

• The course is aimed to help graduate students and advanced software engineers to deeply understand multi-core architecture, study multi-core software development methodology, models, tools and environment.
Content of the Course

Win32 Multithreaded Programming
- Multithreading in the Win32 API
- Thread synchronization

Linux Multithreaded Programming
- Multithreading in Pthreads
- Thread synchronization
- GDB

Introduction to Multi-core
- Hardware
- System Software
- Application Software

Introduction to Parallel Computing
- Hardware Platform
- Computational Models
- Programming Models
- Programming Environment
- Programming Tools

Lab & Mini-project
- Integrated Practice
  - Learn by Doing
  - Project-driven Learning

OpenMP Programming
- Data dependencies
- Synchronization
- Performance

MPI Programming
- Message Passing
- Communicator
- Group
- Topology

Software Development Tools
- C++ Compiler
- VTune Performance Analyzer
- MKL (Math kernel library)
- Thread Checker
- Thread Profiler

Lab & Mini-project
Introduction to Multi-core Technology

- Multi-Processors, Multi-Computer, and Multi-core Microprocessor (0.5hr)
- Multi-core Computer Architecture (1hr)
  - Intel Multi-core architecture introduction
  - Memory and Cache organization
  - Multi-core interconnection
  - Multi-core System on Chip
  - Intel Multi-core Chipset
- OS and BIOS support to multi-core microprocessor (1hr)
  - Trends in multi-core platforms
  - Scheduling policy and algorithms
  - Synchronization and deadlocks
  - Operating System Design Principles for Multi-core
- Virtualization Technology support to multi-core microprocessor (0.5hr)
Introduction to Parallel Computing

• Parallel Computing Theory (3hrs)
  – Parallel Programming model
  – Parallel Algorithm

• Parallel Development Environment (3hrs)
  – Parallel Programming Language
  – Parallel Virtual Machine
  – Parallel Compiler
  – Multitask
  – Multi-process
  – Multi-thread
Network and MPI Programming

- Network Programming (9hrs)
  - Introduction to TCP/IP and Internet
  - Introduction to Network Programming
  - TCP/IP network Programming Framework
  - Network Programming Case Study
- MPI Programming and Tuning (6hrs)
  - Introduction to MPI
  - Install and configure MPICH
  - MPI Programming
  - MPI Cluster Communication
  - MPI Performance Analysis and Tuning
Multithread Programming

- **Windows Multithread Programming** (6hrs)
  - Introduction to Win32 Thread Library
  - Multithreading in Win32 API
  - Understanding Thread executing and Resource Access
  - Multithread Programming and Tuning

- **Linux Multithread Programming** (6hrs)
  - Introduction to POSIX Pthreads Library
  - Multithreading in Pthreads
  - Understanding Thread Synchronization
  - Use GDB in Performance Tuning

- **OpenMP Multithread Programming** (6hrs)
  - Introduction to OpenMP
  - Multithreading in OpenMP
  - Understanding Data dependencies and Thread Synchronization
  - OpenMP Programming and Performance Tuning
Multi-core Software Development Tools (6hrs)

• C++ Compiler
• Performance tuning and monitoring tools, such as Intel VTune
  – Tuning for high performance
  – Tuning for low power
  – Tradeoff between performance and power
  – Tradeoff between one-core and Multi-core
• MKL Library
• Intel Multimedia Library IPP
• Runtime Analysis
  – Intel Thread Checker
  – Intel Thread Profiler
• Open source tools such as GCC, GDB
Lab of the Course

- Lab.1  Software Development Tools for Multi-core
- Lab.2.1 Programming and Tuning with Windows Threads
- Lab.2.2 Programming and Tuning with Linux Threads
- Lab.3  Programming and Tuning with OpenMP
- Lab.4  Programming with MPI
- Lab.5.1 Correcting Threading Errors with Intel® Thread Checker
- Lab.5.2 Tuning Threaded code with Intel® Profiler
Lab.1 Multi-core Software Development Tools

- Intel compiler switches.
- Intel Vtune Performance Analyzer.
- IPP.
- Intel Thread Checker.
- Intel Thread Profiler.
- open source tools.
VTune性能分析器

- 调用曲线图
  - 分析程序运行时函数的入口点和出口点
  - 生成一张调用曲线图并且确定调用顺序和显示关键路径

- 计数器监控器
  - 在运行时跟踪系统活动

- 调优助手
  - 根据丰富的知识库分辨性能问题
  - 自动推荐代码改进办法
Thread Checker线程检查器的使用

寻找潜在的数据竞争

编译和运行Potential程序的单线程版本
• 打开\code\Thread Checker\potential_serial文件夹，双击potential_serial.sln文件；
• 在Build菜单里选择Configuration Manager，然后选择Debug模式；
• 在Build菜单里选择Build Solution，编译相关文件；
• 在Debug菜单里选择Start Without Debugging，运行程序。

编译和运行Potential程序的多线程版本
• 打开\code\Thread Checker\potential_win文件夹，双击potential_win.sln文件；
• 在Build菜单里选择Configuration Manager，然后选择Debug模式；
• 按如下方式配置项目属性：
  – 选中Debug模式（/Zi）
  – 链接时保留Debug信息（/DEBUG）
  – 禁止自动优化（/Od）
  – 使用线程安全系统库（/MDd）
  – 使用二进制文件可重定位功能（/fixed:no）
• 在Build菜单里选择Build Solution，编译相关文件；
• 在Debug菜单里选择Start Without Debugging，运行程序；
• 运行英特尔VTune性能分析器；
• 点击New Project；
• 在Category栏选择Threading Wizards，在下拉框中选择Intel Thread Checker Wizard；
• 选择刚才编译好的可执行文件路径（\code\Thread Checker\potential_win\Debug\potential_win.exe），点击Finish按钮，开始运行线程检查器；
• 线程检查器分析完毕后会显示一些诊断报告，双击可观察相应代码；
Thread Profiler线程档案器的使用

- 打开`\code\ThreadProfiler\potential lab 1`文件夹，双击Potential Lab 1.sln文件；
- 在Build菜单里选择Configuration Manager，然后选择Release模式；
- 按如下方式配置项目属性：
  - 选中Debug模式（/Zi）
  - 链接时保留Debug信息（/DEBUG）
  - 使用线程安全系统库（/MD）
  - 使用二进制文件可重定位功能（/fixed:no）
• 在Build菜单里选择Build Solution，编译相关文件；
• 运行英特尔VTune性能分析器；
• 点击New Project；
• 在Category栏选择Threading Wizards，在下拉框中选择Intel Thread Profiler Wizard；
• 选择刚才编译好的可执行文件路径（\code\Thread Profiler\Potential lab 1\Release\potential.exe），点击Finish按钮，开始运行线程档案器；
• 运行结束后，可看到双重视图
分析程序

• 返回到Profile栏的Concurrency Level视图，有多少时间消耗在串行（只有一个线程）执行程序？多少时间消耗在完全的并行执行？从这能得到什么启发？
• 打开grouping to thread视图，该视图显示了程序中所有线程在关键路径上的活跃程度。该程序一共运行过多少个线程？有什么性能问题？
• 打开grouping to object视图，是否有同步对象在关键路径占了相当大比例，而它又是串行执行的？如果有，是哪一个？
• 双击Timeline栏，打开时间轴视图。从这些数据中所发现最明显的特征是什么？如果有性能问题，应该从哪方面去解决它？
Lab.2.1 Programming with Windows Threads

- Practice multi-threaded programming of applications using Win32 threading API.
- Find and resolve a common data race.
- Convert a serial application to a threaded version.
- Find simple data races in code and resolve these threading errors using critical sections as the synchronization mechanism.

HANDLE CreateEvent( … );
HANDLE OpenEvent ( … );

ResetEvent( )
SetEvent ( )
WaitForMultipleObjects ( )
实验目的

• 掌握利用Win32 API进行线程的同步
• 掌握利用MFC库进行线程的同步
• 掌握多线程调试技术
```cpp
#include "stdafx.h"
#include <windows.h>
#include <process.h>
#include <iostream>
#include <fstream>
using namespace std;
HANDLE evRead, evFinish;
void ReadThread(LPVOID param)
{
    WaitForSingleObject (evRead ,INFINITE);
    cout<<"Reading"<<endl;
    SetEvent (evFinish);
}
void WriteThread(LPVOID param)
{
    cout<<"Writing"<<endl;
    SetEvent (evRead);
}
```
int main(int argc, char * argv[])
{
    evRead = CreateEvent(NULL, FALSE, FALSE, NULL);
    evFin = CreateEvent(NULL, FALSE, FALSE, NULL);
    _beginthread(ReadThread, 0, NULL);
    _beginthread(WriteThread, 0, NULL);
    WaitForSingleObject(evFinish, INFINITE);
    cout << "The Program is End" << endl;
    return 0;
}

程序输出如下:
Writing
Reading.
The Program is End
Lab.2.2 Programming with Linux Threads

- Practice multi-threaded programming of applications using Pthreads.
- Find and resolve a common data race.
- Convert a serial application to a threaded version.
- Find simple data races in code and resolve these threading errors using critical sections as the synchronization mechanism.

Create Thread: `pthread_create()`
Thread exit:
  `pthread_exit()`, `pthread_cancel()`
Waiting the end of Thread: `pthread_join()`
Thread detach: `pthread_detach()`
```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
#define THREAD_NUMBER 2
int retval_hello1= 2, retval_hello2 = 3;
void* hello1(void *arg)
{
    char *hello_str = (char *)arg;
    sleep(1);
    printf("%s\n", hello_str);
    pthread_exit(&retval_hello1);
}
void* hello2(void *arg)
{
    char *hello_str = (char *)arg;
    sleep(2);
    printf("%s\n", hello_str);
    pthread_exit(&retval_hello2);
}
```

Output:
```
pku@pku-laptop:~$ ./helloworld
Begin to create threads...
Begin to wait for threads...
hello world from thread1
return value is 2
hello world from thread2
return value is 3
Now, the main thread returns
```
int main(int argc, char *argv[]) {
    int i;    int ret_val;    int *retval_hello[2];
    pthread_t pt[THREAD_NUMBER];    const char *arg[THREAD_NUMBER];
    arg[0] = "hello world from thread1";
    arg[1] = "hello world from thread2";
    printf("Begin to create threads...
");

    ret_val = pthread_create(&pt[0], NULL, hello1, (void *)arg[0]);
    if (ret_val != 0 ) { printf("pthread_create error!
"); exit(1); }
    ret_val = pthread_create(&pt[1], NULL, hello2, (void *)arg[1]);
    if (ret_val != 0 ) { printf("pthread_create error!
"); exit(1); }
    printf("Now, the main thread returns.
");
    printf("Begin to wait for threads...
");
    for(i = 0; i < THREAD_NUMBER; i++) {
        ret_val = pthread_join(pt[i], (void **)&retval_hello[i]);
        if (ret_val != 0) { printf("pthread_join error!
"); exit(1); }
        else printf("return value is %d\n", *retval_hello[i]);
    }
    return 0;
}
Lab.3 Threading Methodology using OpenMP

- Use the most common OpenMP C statements.
- Compile and run an OpenMP program.

```c
#pragma omp parallel for [clause[clause...]]
  for (index = first; test_expr; incr_expr) {
    body of the loop;
  }

//instance:
#pragma omp parallel for
for (int i=0; i<n; i++)
  z[i]=x[i]+y[i];

#pragma omp parallel [clause[clause]...]
  block

//instance:
#pragma omp parallel
#pragma omp for
for (int i=0; i<5; i++)
  printf("hello world i=%d\n",i);
```
实验目的

- 掌握基于OpenMP多线程应用程序开发
- 掌握基于OpenMP多线程同步
- 掌握基于OpenMP多线程应用程序性能分析
循环并行化

```c
#pragma omp parallel for [clause[clause…]]
    for (index = first; test_expr; incr_expr)
    {
        body of the loop;
    }
```

实例:
```c
#pragma omp parallel for
for (int i=0;i<n;i++)
    z[i]=x[i]+y[i];
```
并行区域编程

#pragma omp parallel [clause[clause]...] block
实例：
#pragma omp parallel
  for (int i=0;i<5;i++)
    printf("hello world i=%d\n",i);
int i; int max_num_x=max_num_y=-1;
#pragma omp parallel for
for (i=0;i<n;i++)
{
#pragma omp critical (max_arx)
    if (arx[i]>max_num_x)
        max_num_x=arx[i];
#pragma omp critical (max_ary)
    if (ary[i]>max_num_y)
        max_num_y=ary[i];
}
原子操作

```c
int counter=0;
#pragma omp parallel
{
    for (int i=0;i<10000;i++)
        #pragma omp atomic  //atomic operation
        counter++;
}
printf("counter = %d\n",counter);
```
Lab.4 Programming with MPI

- **MPI_Init:**
  - initialize MPI environment
- **MPI_Finalize:**
  - Finalize MPI environment
- **MPI_Comm_rank:**
  - Mark MPI Processes
- **MPI_Comm_size:**
  - Mark the numbers of processes in a MPI Process group
**MPI 程序的四个基本函数**

- **MPI_Init:**
  - 初始化 MPI 执行环境

- **MPI_Finalize:**
  - 结束 MPI 执行环境

- **MPI_Comm_rank:**
  - 标识各个 MPI 进程的

- **MPI_Comm_size:**
  - 用来标识相应进程组中有多少个进程
```c
#include <stdio.h>
#include "mpi.h"
int main ( int argc, char *argv[] ) {
    int rank;
    int size;
    MPI_Init ( argc, argv );
    MPI_Comm_rank ( MPI_COMM_WORLD, &rank );
    MPI_Comm_size ( MPI_COMM_WORLD, &size );
    printf ( "Hello world from process %d of %d\n", rank, size );
    MPI_Finalize ( );
    return 0;
}

Output:
Hello world from process 0 of 4
Hello world from process 1 of 4
Hello world from process 2 of 4
Hello world from process 3 of 4
```
Lab.5 Advanced Tools

- Lab.5.1 Correcting Threading Errors with Intel Thread Checker
  - Debug Win32 threaded applications and resolve data race conditions.
  - Use the advanced features of Intel Thread Checker.
- Lab.5.2 Tuning Threaded code with Intel Profiler
  - Use Intel Thread Profiler to detect performance issues in applications threaded with the Win32 threading API.
  - Navigate through the Thread Profiler features to understand program thread activity.
Intel Thread Checker

- Plug-in to VTune™ Performance Analyzer
- Debugging tool for threaded software
- Locates bugs quickly
  - Data races or storage conflicts
    - More than one thread accesses memory without synchronization
  - Deadlocks
    - Thread waits for an event that will never happen
Starting Thread Checker

Results
Intel Thread Profiler

• Plugs in to the VTune™ performance environment
• Pinpoints performance bottlenecks that directly affect execution time
• Identifies performance issues in OpenMP* or threaded applications using the Win32* API and POSIX* threads
Setting Compile & Link Options

- Selected the Release build
- Make sure that Debug format is specified
- Make sure that the Debug symbols are generated for the application
- Make sure that thread safe libraries have been selected
- Make sure that application is built with /fixed: no option from linker>
- Advanced attribute
Creating a New Project a Thread Profiler Activity

- Start Vtune Performance Analyzer and select new project
- From the Category Threading Wizards, select the Intel Thread Profiler Wizard.
- Be sure to click on the “Thread(Windows* API or POSIX* threads)” radio button is selected
- Select “Thread Profiler” and click on “configure”
- Select the “Thread Activity” and “Transitions” check boxes. This selection will issue a warning that the Thread Profiler may take longer to run. Accept this warning by clicking on “yes”.

![Image of Vtune Performance Analyzer Easy Start window]
Mini-project of the Course

• Select one of 20 mini Projects in order to develop a multi-core application, such as: MP4 Converter, Streaming Server, Mail Server, ftp Server etc.
Mini-project 1: MP4 Converter

- Convert AVI, MPEG and MP4 video to H.264 video

Reference: Eric Shufro “Video Transcoding with Intel IPP”
http://www.cse.fau.edu/~hari/courses/2004/VidCom/eric_shufro_project.ppt
Intel® Integrated Performance Primitives

- Providing source code and libraries for media types such as MP3, MPEG-2, MPEG-4, H.263, H.264, JPEG, JPEG2000 etc.
- Well documented.
- Easy to use.

- Transcorder Architecture
  - Decoder and encoder based on the IPP.
  - Transcorder class encapsulates both the encoder and decoder.
  - Memory is accessible between the encoder and the decoder.
  - Transcorder runs in three separate threads.
Mini-project 2: Porting open source streaming server

- Project: Port an open source streaming server to Intel Multi-core Platform
  - Port Darwin Streaming Server to multi-core platform.
  - Optimize Darwin Streaming Server.
  - Real-time streaming scheduling algorithms
• Q&A

• Thank you!