

OpenMP in aller Kürze

Dipl.-Inf. Domenic Jenz
(jenz@hirs.de)

HLRS, Universität Stuttgart

Übungen zur Vorlesung Molekulardynamik und Lattice
Boltzmann Methoden

Outline

Zugang BW-Grid

- Zertifikat beantragen
- Zertifikate vorbereiten
- gsi-ssh

OpenMP

- Einführung
- Arbeitsteilung
- Gültigkeitsbereiche
- Synchronisation
- kleine Aufgabe



Übungstermine (geplant)

28.10.2008

11.11.2008

25.11.2008

9.12.2008

13.1.2009

27.1.2009

10.2.2009



BW-Grid Daten



- ▶ 498 Knoten
- ▶ Pro Knoten: 2x Intel Xeon E5440 "Harpertown" (Quadcore), 2.84 GHz, 12MB Cache, 16 GB RAM
- ▶ Platz 65 der Top 500 (Stand 06/08)
- ▶ vom ehemaligen D-Grid Cluster: u.a. 7 Cell Knoten



Zertifikat beantragen

siehe auch unter <http://www.hlrs.de/people/niebling/bwgrid>

- ▶ https://pki.pca.dfn.de/grid-root-ca/cgi-bin/pub/pki?cmd=getStaticPage&name=index&RA_ID=123
anwählen
- ▶ unter **Nutzerzertifikat** den Antrag ausfüllen und ausdrucken.
- ▶ mit dem Ausdruck und Personalausweis zur Registrierungsstelle gehen
- ▶ Nach ein paar Tagen sollte eine Mail mit dem Link zum CA-Zertifikat und zum eigenen Zertifikat kommen. Das Zertifikat kann dann in den Browser mit dem man den Antrag gestellt hat, importiert werden.



Mitgliedschaft bei einer VO

Um jetzt mit dem Zertifikat die Ressourcen auch nutzen zu können, muss man sich noch bei einer virtuellen Organisation anmelden.

- ▶ Mit dem Browser, der das Zertifikat geladen hat, **<https://dispatch.fz-juelich.de:8814/D-Grid-VO-Member>** aufrufen.
- ▶ In der Zeile mit bwgrid, den Link **Member Registration** anwählen und dort den Antrag stellen.
- ▶ Nach Antragsstellung wird eine E-Mail zur Bestätigung der eigenen E-Mail Adresse zugesandt. Diese muss innerhalb von 10 Tagen bestätigt werden
- ▶ Nach ein paar Tagen ist dann hoffentlich der Antrag angenommen (status unter Member Info ist dann "approved")



Eigenes Zertifikat

- ▶ sollte schon gemacht worden sein: Backup des DFN Zertifikats aus dem Browser im PKCS12 Format
- ▶ Für gsi-ssh werden muss dann das Zertifikat noch aufgeteilt werden:
 - ▶ **openssl pkcs12 -in zertifikat.p12 -out userkey.pem -nocerts** (Das erste Passwort ist das Backup-Passwort)
 - ▶ **openssl pkcs12 -in zertifikat.p12 -out usercert.pem -clcerts -nokeys**
- ▶ Die beiden Dateien userkey.pem und usercert.pem dann in das Verzeichnis **.globus** im eigenen Heimverzeichnis verschieben.



gsi-ssh 1

- ▶ Zum einloggen verwenden wir nun GSI-SSHterm:
http://www.grid-support.ac.uk/content/view/81/61/

The screenshot shows a web browser window with the address bar containing the URL <http://www.grid-support.ac.uk/content/view/81/61/>. The page content is for the 'GSI-SSHterm Application' on the NGS (National Grid Service) website. The page features a navigation menu with links for Home, Services, Labs, Users, Documentation, and Support. Below the navigation is a search bar. The main content area contains the following text:

GSI-SSHterm Application

GSI-SSHterm is a desktop application, maintained by the NGS, designed to allow you to easily and securely access the Grid using GSISSH. By default it should work with any UK e-Science Grid where you have an account (e.g. the NGS) using your UK e-Science certificate or a proxy certificate stored in the MyProxy server. For more details click [here](#) or read the [FAQ](#).

For more detailed on advanced uses of the GSI-SSHterm and re-distributing it for our Grid see the [NGS wiki](#), and to use the GSI-SSHterm as a web applet click [here](#).

Use Java Web Start to load the terminal

An easy and straight forward method for using the GSI-SSH Terminal is to use Java Web Start. This automatically installs the terminal as an application on your machine and provides a shortcut on your desktop.

To install and run the GSI-SSH Terminal on your machine please [click install!](#)

Problems running the terminal?

Please first check the [FAQ](#). If you still have problems, contact the [helpdesk](#).

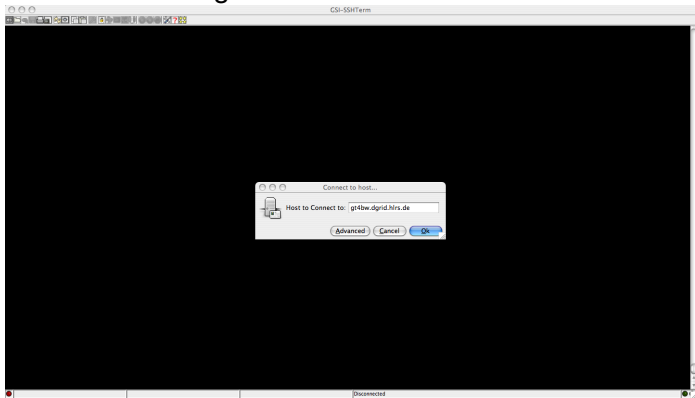
GSI-SSHterm was originally developed by NRC and enhanced by the NGS.

Last updated (Monday, 13 October 2008)

- ▶ Die Datei dann mit javaws öffnen.

gsi-ssh 2

- ▶ Neue Verbindung über File->New Connection anfordern



- ▶ Der Host zu dem wir uns verbinden ist **gt4bw.dgrid.hlr.de**

Auf BW-Grid

- ▶ Nun sind wir auf einem Frontend des BW-Grids
- ▶ Zum Rechnen holen wir uns einen Knoten (8 Cores):
qsub -VIX -lnodes=8, walltime=3:00:00
- ▶ Auf diesem Knoten brauchen wir noch einen Compiler, der OpenMP beherrscht (der gcc dort ist zu alt):
modules load compiler/intel
Damit sollte der Intel 10.1 Compiler über **icc** bzw. **icpc** aufrufbar sein.
- ▶ An Editoren gibt es reichlich Auswahl: u.a. emacs, vim, kate, nano, ed
- ▶ Zum Kompilieren mit OpenMP:
icc -openmp -o Ausgabe Eingabe.c
gcc -fopenmp -o Ausgabe Eingabe.c



OpenMP Einleitung

- ▶ OpenMP ist eine standardisierte Programmierschnittstelle (API) für shared memory Systeme
- ▶ Ermöglicht es auf einfache Art und Weise multi-threaded zu programmieren, indem man Compilerdirektiven hinzufügt
- ▶ Bindings existieren für C, C++ und Fortran
- ▶ Eignet sich vor allem zur Parallelisierung von for-Schleifen
- ▶ OpenMP gestattet eine inkrementelle Parallelisierung



Parallele Regionen

Mit **#pragma omp parallel [parameter...]** wird der darauffolgende Block auf die Threads aufgeteilt und in jedem Thread separat ausgeführt. Das einfachste Beispiel:

```
1 #include <omp.h>
2 #include <stdio.h>
3
4 int main ()
5 {
6     #pragma omp parallel
7     {
8         printf ("Hello_World_\n");
9     }
10    return 0;
11 }
```

Wie viele denn jetzt ?

Jetzt sollte für jeden Thread **Hello World** ausgegeben worden sein.

Aber wie kann man beeinflussen, wie viele Threads erzeugt werden ? In Präzedenzreihenfolge:

- ▶ Setzen des **NUM_THREADS** Parameters im **#pragma omp parallel**
- ▶ Über die Funktion **omp_set_num_threads(int)**
- ▶ Setzen der Umgebungsvariable **OMP_NUM_THREADS** z.B. mit **export OMP_NUM_THREADS=5**
- ▶ Implementationsabhängige Voreinstellung, üblicherweise die Anzahl der CPUs/Kerne auf einem Knoten



Wer bin ich ?

Oft ist es auch ganz praktisch zu wissen wie viele Threads laufen und welcher man gerade selbst ist:

```
1 #pragma omp parallel
2 {
3     printf ("I_am_%d_of_%d\n", omp_get_thread_num(),
4         omp_get_num_threads ());
5 }
```

Hier gibt jetzt jeder Thread seine eigene Nummer (Zählung beginnt mit 0) aus und die Gesamtzahl der Threads.



Parallelisieren von for-Schleifen

Eine for-Schleife innerhalb einer parallelen Region kann auf die Threads verteilt werden mit Hilfe von **#pragma omp for [Parameter...]**:

```
1 int i, a[N], b[N], c[N];
2 // Initialisieren von a,b,c ...
3 #pragma omp parallel
4 {
5     #pragma omp for
6     for (i = 0; i < N, i++) {
7         printf ("%d_is_at_%d:\n", i,
8                 omp_get_thread_num ());
9         a[i] = b[i] + c[i];
10    }
11 }
```

Hier werden die Iterationen auf die einzelnen Threads aufgeteilt und parallel ausgeführt.

Beeinflussung der Aufteilung

Die Aufteilung der Iterationen bei **#pragma omp for [Parameter..]** lässt sich über Parameter beeinflussen:

- ▶ **schedule (type, [chunk]):**
type kann dabei u.a. folgendes sein:
 - ▶ **STATIC**
die Schleifeniterationen werden in Pakete der Grösse *chunk* eingeteilt und dann statisch an die Threads verteilt. Wenn *chunk* nicht angegeben wurde, werden die Iterationen, so denn möglich, gleichmässig verteilt.
 - ▶ **DYNAMIC**
die Schleifeniterationen werden in Pakete der Grösse *chunk* eingeteilt und dynamisch an die Threads vergeben. Wenn ein Thread mit seinem Paket fertig ist, bekommt er ein neues. Die *chunk*-size ist, wenn nicht angegeben, 1.

Sektionen

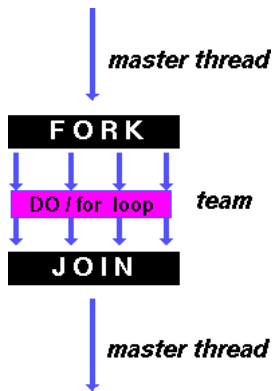
Etwas allgemeiner sind Sektionen:

```
1 #pragma omp parallel
2 {
3     #pragma omp sections
4     {
5         #pragma omp section
6         {
7             a();
8         }
9         #pragma omp section
10        {
11            b();
12        }
13    }
14 }
```

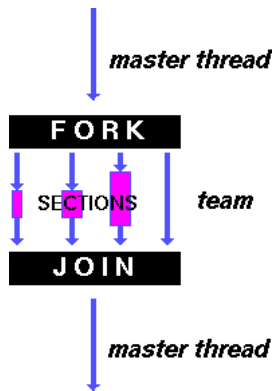
a() und b() werden hier parallel ausgeführt.



Ein Bild dazu



(a) paralleles for



(b) Sektionen

Beispiel

```
1 int wichtig = 42;
2 int id;
3 #pragma omp parallel private(id) shared(wichtig)
4 {
5     int h = 0, i;
6     printf ("`id zuvor: %d\n'", id);
7     id = omp_get_thread_num ();
8     for (i = 0; i < wichtig; i++) {
9         h += i;
10    }
11    printf ("`id: %d, h = %d\n'", id, h);
12 }
```

Was passiert wenn **id** auf shared gesetzt wird ?

weitere Gültigkeitsbereiche

Es gibt noch mehr als **shared** und **private**:

- ▶ **firstprivate(Variablenliste)**

Wie **private**. Die Werte werden aber mit dem Originalwert der Variable vor dem parallelen Bereich initialisiert

- ▶ **lastprivate(Variablenliste)**

Kann nur bei **omp for** und **omp sections** verwendet werden. Damit wird der Wert aus dem letzten Schleifendurchlauf bzw. aus der letzten Sektion in die Originalvariable geschrieben.

- ▶ **reduction(Operator:Variablenliste)**

Für jeden Thread wird eine private Kopie der Variablen erstellt und am Ende werden diese Kopien anhand des angegebenen Operators (z.B. +, *,) verknüpft und das Ergebnis schliesslich in die Originalvariable kopiert.



Was schief gehen kann...

```
1 #include <omp.h>
2 #include <stdio.h>
3
4 int main ()
5 {
6     double h = 0.0;
7     int i = 0;
8     #pragma omp parallel for shared(h) private(i)
9     for (i = 1; i < 1000; i++)
10    {
11        h += 1 / (double) i;
12    }
13    sleep (1);
14    printf (``h is %f\n'', h);
15 }
```

Was schief gehen kann...

Manche Anweisungen sollten nicht parallel ausgeführt werden. Um dies zu verhindern gibt es u.a.:

- ▶ **#pragma omp master**

Der nachfolgende Block wird nur vom Masterthread (0) ausgeführt.

- ▶ **#pragma omp critical**

Der nachfolgende Block wird zu jedem Zeitpunkt nur von max. einem Thread ausgeführt. Wenn ein Thread in einer critical Region ist und ein anderer Thread dieselbe Region erreicht, muss dieser warten, bis der erste Thread fertig ist.



Weiterführendes

Dies war jetzt nur ein kurzer unvollständiger Überblick über OpenMP.

- ▶ Zur Weiterbildung sind die HLRS Kurse sehr zu empfehlen:
Unter <http://www.hlrs.de/news-events/events/> gibt es eine Übersicht der Kurse. Darunter eben auch MPI und OpenMP Kurse.
- ▶ Die Materialien gibt es auch online:
http://www.hlrs.de/organization/par/par_prog_ws/
- ▶ Eine nette Übersicht der OpenMP Befehle gibt es unter:
<https://computing.llnl.gov/tutorials/openMP/>

kleine Aufgabe 1

- ▶ Ladet von <http://www.hlr.de/people/jenz> die Datei `pi.c` runter.
- ▶ Diese enthält ein kleines serielles Programm zur Berechnung von π
- ▶ Kompiliert das Programm zuerst ohne OpenMP mit **`gcc -o pi-seriell pi.c`** oder **`icc -o pi-seriell pi.c`** und lasst es laufen
- ▶ Kompiliert das Programm mit OpenMP **`gcc -fopenmp -o pi pi.c`** bzw. **`icc -openmp -o pi pi.c`** und lasst das auch mal laufen, nachdem ihr **`export OMP_NUM_THREADS=4`** ausgeführt habt.
- ▶ sollte eigentlich keine grossen Unterschiede geben.



