TECHNISCHE UNIVERSITÄT DRESDEN

Center for Information Services and High Performance Computing (ZIH)

Programmspuren

VampirTrace und Vampir

07. Januar 2010

INF 1038
Nöthnitzer Straße 46
01187 Dresden
0351 - 463 38474

Thomas William

ZIH
Center for Information Services & High Performance Computing

Foliensatz

Verfügbarkeit der Folien

Vorlesungswebseite:
`http://tu-dresden.de/die_tu_dresden/zentrale_einrichtungen/zih/lehre/ws0910/lctp`

Inhalt

Instrumentieren und Tracen

Allgemeine Betrachtung

Tracing - Programmspuren

Definition:

Recording of run-time events (points of interest)

- Zur Laufzeit des Programms
- Anfang / Ende von Funktionen / Subroutinen
- Senden / Empfangen von Nachrichten
- Synchronisationspunkte
- ...
- Eintrag / Datensatz pro Ereignis
- Zeitstempel, ProzessID, Thread, Ereignistyp
- Nach Zeitstempel sortiert

Daten werden meist durch Bibliothek gesammelt und gespeichert
$\Longrightarrow$ "instrumentation & trace library"

Parallele Programmspur

```
10010 P 1 ENTER 5
10090 P 1 ENTER 6
10110 P 1 ENTER 12
10110 P 1 SEND TO 3 LEN 1024 ...
10330 P 1 LEAVE 12
10400 P 1 LEAVE 6
10520 P 1 ENTER 9
10550 P 1 LEAVE 9
...
```

```
10020 P 2 ENTER 5
10095 P 2 ENTER 6
10120 P 2 ENTER 13
10300 P 2 RECV FROM 3 LEN 1024 ...
10350 P 2 LEAVE 13
10450 P 2 LEAVE 6
10620 P 2 ENTER 9
10650 P 2 LEAVE 9
...
```

```
DEF TIMERRES 1000000000
DEF PROCESS 1 `Master`
DEF PROCESS 1 `Slave`
DEF FUNCTION 5 `main`
DEF FUNCTION 6 `foo`
DEF FUNCTION 9 `bar`
DEF FUNCTION 12 `MPI_Send`
DEF FUNCTION 13 `MPI_Recv`
```


Profiling versus Tracing

Vorteile Tracing

- Zeitlicher und räumlicher Zusammenhang bleibt erhalten
- Dynamisches Verhalten in beliebiger Genauigkeit abbildbar
- Statistiken / Profile können aus der Programmspur abgeleitet werden

Nachteile Tracing

- Programmspuren können sehr groß werden
- Belastung / Störung des Programmablaufs größer als beim Profiling
- Instrumentierung / Tracing kompliziert
- Probleme beim Zwischenspeichern der Werte
- Uhrensynchronisation . . .

Ereignis Typen

Anfang / Ende einer Funktion (Region etc.)

```
time stamp, Prozess/thread, function ID
```

Senden / Empfangen einer Nachricht (P2P, MPI)

```
time stamp, sender, receiver, length, tag, communicator
```

Kollektive Kommunikation

```
time stamp, Prozess, root, communicator, #bytes
```

Hardware Performance Counter Werte

```
time stamp, Prozess, counter ID, value
```

. . .

Instrumentierung

Instrumentation:

Prozess of modifying programs to detect and report events by calling instrumentation functions.

- Meist werden die *instrumentation functions* von einer Bibliothek bereit gestellt
- Diese erzeugen Nachrichten beim Eintreten bestimmter Ereignisse zur Laufzeit
- Es gibt mehrere Möglichkeiten ein Programm zu instrumentieren

Quelltext Instrumentierung

Original

```
1 int foo(void* arg){
2
3 if (cond){
4
5 return 1;
6 }
7
8 return 0;
9 }
```

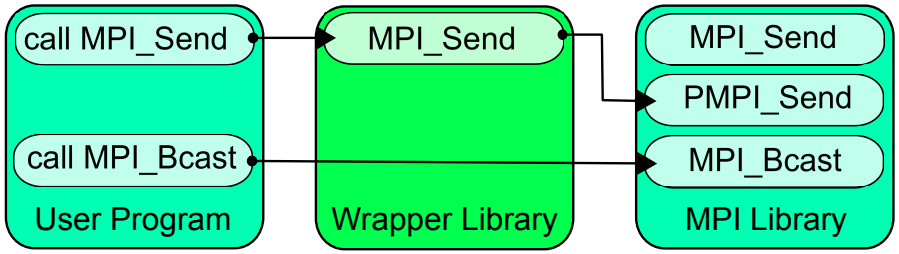
Instrumentiert

```
1 int foo(void* arg){
2 enter(7);
3 if (cond){
4 leave(7);
5 return 1;
6 }
7 leave(7);
8 return 0;
9 }
```

Die Instrumentierung kann manuell oder automatisch erfolgen.

Die *MPI Profiling* Schnittstelle

- Zu jeder MPI-Funktion gibt es 2 Namen:
  - `MPI_xxx` und `PMPI_xxx`
- Die MPI-Funktionen können zur Laufzeit selektiv ersetzt werden



Manuelle und automatische Instrumentierung

Manuell

- Großer Aufwand
- Fehleranfällig
- Schwer zu handhaben

Automatisch

- Quelltext-zu-Quelltext Transformation
- Compiler
- Vorladen eine Bibliothek (siehe LD_PRELOAD)
- Dynamisch zur Laufzeit

OTF und Vampirtrace

Beispiele praktischer Implementierungen

OTF - Open Trace Format - Kurzübersicht

- Open Source Trace File Format (OTF)
  - Entwickelt an der Technische Universität Dresden, ZIH
  - Bibliothek zum Schreiben paralleler Traces
  - In C geschrieben, unter der BSD verfügbar
  - `http://www.tu-dresden.de/zih/otf`
- Eigenschaften
  - Menschen lesbare Programmspuren
  - Schnelle Suche und Indizierung
  - Snapshots
  - On-the-fly Kompression
- Aktive Entwicklung
  - In Kooperation mit der Universität von Oregon (TAU) und dem
  - Lawrence Livermore National Laboratory

VampirTrace - Kurzübersicht

- Open Source "Programm Monitor"
  - Entwicklung der Technischen Universität Dresden, ZIH
  - Teil von OpenMPI (seit Version 1.3)
  - (Beta-)Support für Cell/B.E. und NVIDIA Cuda
  - `http://www.tu-dresden.de/zih/vampirtrace`
- Aufgezeichnete Ereignisse
  - Funktion Anfang/Ende
  - MPI and OpenMP Ereignisse
  - Hardware/Software performance counters (zum Beispiel PAPI)
  - Ereignisse im Betriebssystem: Prozess Erzeugung, Ressourcenverwaltung
- Eigenschaften der aufgezeichneten Werte
  - Zeitstempel
  - Ort (Prozess / Thread / MPI)
  - MPI spezifische Werte wie Nachrichtengröße etc.

VampirTrace

- Versteckt den Vorgang des Instrumentierens in einem *smart-script*
- Alles Relevante wird für den Nutzer transparent gehandhabt
- Im Makefile muss lediglich der Compiler-Aufruf ersetzt werden:
  - `CC=mpicc` $\Longrightarrow$ `CC=vtcc -vt:cc mpicc`
- C++ and Fortran: `vtcxx, vtf77, vtf90`
- Die für Einsteiger wichtigsten Flags sind:
  - `vtcc -vt:help`
  - `vtcc -vt:showme`
  - `vtcc -vt:verbose`

Übersicht der wichtigsten Eigenschaften von VampirTrace

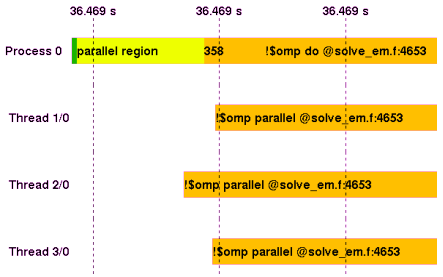
- Unterstützung für MPI und OpenMP
- Automatisches Instrumentieren von Funktionen durch viele Compiler
- Unterstützung der *Hardware Performance Counter* (PAPI)
- Optional aufzeichnen der Speicherallokations-Aufrufe
- E/A Unterstützung (Funktion, Dateiname, Anzahl der Bytes)
- Manuelle Instrumentierung des Quelltextes mit Hilfe von Makros
- Nutzerdefinierte Zähler (z. Bsp. Wert der Schleifen-Zählvariable)
- Laufzeitfilter und Gruppierung der Funktionen zur besseren Übersicht
- Nachträgliches filtern per Kommandozeilenprogramm

free: www.tu-dresden.de/zih/vampirtrace

MPI / OpenMP tracing

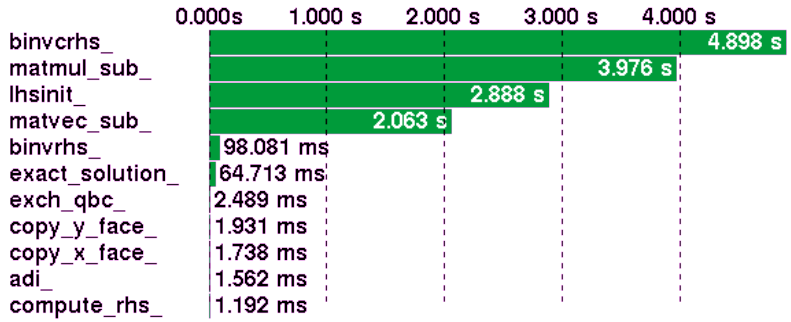
- Tracen von MPI-1 and MPI-IO Ereignissen über die PMPI-Schnittstelle

- Tracen von OpenMP Anweisungen via OPARI Quelltext Transformation



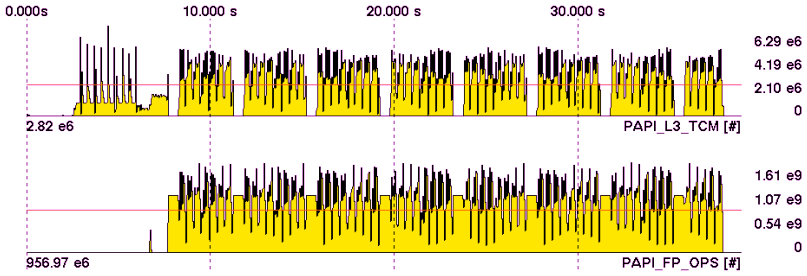
Automatisches Instrumentieren von Funktionen

- Nutzt Compiler-Unterstützung (Flags) um Trace-Anweisung am Anfang / Ende jeder Funktion einzufügen
- Unterstützte Compiler:
  - GNU, Intel 10, PGI, PathScale, IBM, Sun Fortran, NEC
- Instrumentation von Anwendungen (Binärdaten) durch Dyninst



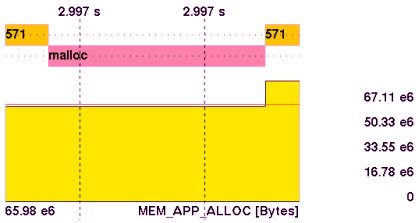
Hardware Performance Counter

- PAPI ist eine Schnittstelle zum Zugriff auf hardware-nahe Register / Zähler (FLOP, Cache Misses)
- Vampirtrace zeichnet die Werte aller abonnierten Zähler am Anfang / Ende jeder Funktion auf

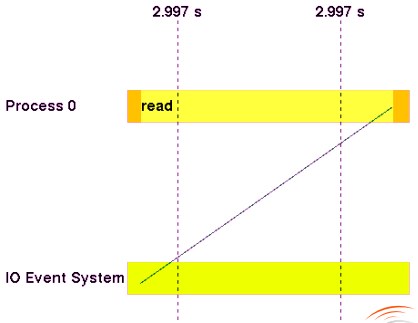


Speicher and E/A Tracing

- Speichertracing erfolgt über sog. "Hooks" in der libc
- `malloc, realloc, free ...`



- Tracing von E/A Aufrufen, Transferraten, Dateizugriffen durch Wrapper
- `open, read, write ...`



Filtern zur Laufzeit

- Sehr kleine (kurze) und häufig genutzte Funktionen führen zu sehr großen Datensätzen
- Selektives Filtern zur Laufzeit behebt dieses Problem
- Dafür wird die Variable `VT_FILTER_SPEC` auf den Pfad zu einer Filterdatei gesetzt:

```
rand;add -- 0
module_utilities_* -- 1000
* -- -1
```

- Die Aufzeichnung für `rand` und `add` wird komplett abgeschaltet
- Alle Funktionen, welche mit `module_utilities` beginnen, werden maximal 1000 mal gelogged
- Alle anderen Funktionen werden unbegrenzt aufgezeichnet (Standard)

Funktionsgruppen

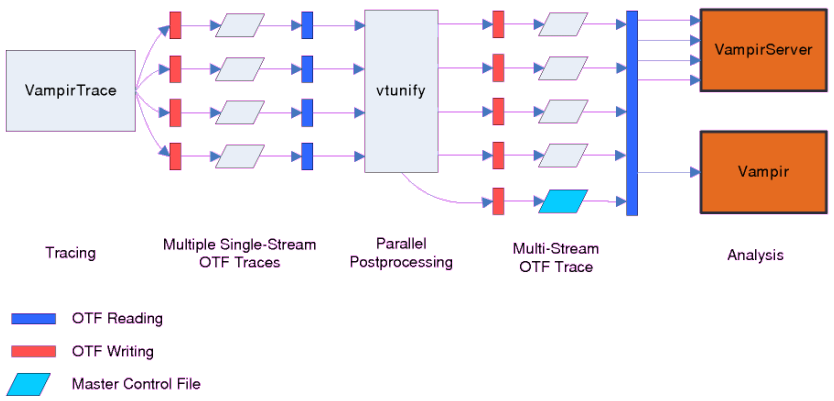
- Funktionen können zu Gruppen zusammengefasst werden
- Gruppen werden für die Visualisierung Farben zugeordnet
- Es existieren folgende Standardgruppen: MPI, OMP, MEM, I/O, Application
- Beispieldatei für Gruppen:

```
CALC=*physics*;*jacobi*;
UTIL=module_utilities_*
COMM=comm_lib_*;par_lib_*
```

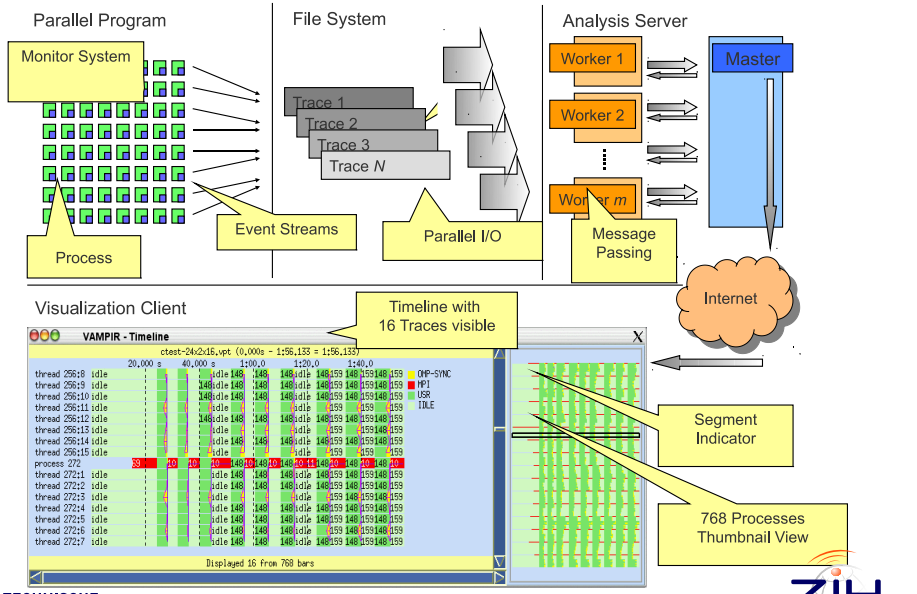

Liste wichtiger Umgebungsvariablen

VT_BUFFER_SIZE	Größe des internen Puffers
VT_MAX_FLUSHES	Maximale Anzahl der Schreibvorgänge
VT_VERBOSE	Ausgabe VampirTrace relevanter Kontrollinformationen
VT_METRICS	Zu messende Counter als Doppelpunkt getrennte Liste
VT_MEMTRACE	Speicher Tracing einschalten
VT_IOTRACE	E/A Tracing einschalten
VT_FILTER_SPEC	Pfad zur Filterdatei
VT_GROUPS_SPEC	Pfad zur Spezifikationsdatei

Verknüpfung VampirTrace, OTF, Vampir



Vampir Next Generation Architektur



Vampir Live Demo

Vampir Demo

- Unter `http://vampir.eu/download/index.html` ist eine Demoversion verfügbar
- Die aktuelle Beta-Version für Linux (i386) befindet sich auch im Ordner xxx

BenchIT

"Contrary to common belief, performance evaluation is an art."
(Raj Jain, 1991)

BenchIT

- BenchIT ist ein Open Source Projekt des ZIH
- Hauptsächlich von Studenten entwickelt
- Auf ANSI-C und POSIX-Shell Skripten basierendes Framework zur Leistungsmessung
- Verschiedene Tests in Form von Kernen zur Ermittlung der Kenndaten des Systems
- Über eine Schnittstelle können einfach weitere Kerne erstellt werden
- Eine GUI unterstützt bei der Auswertung der Ergebnisse
- Per SSH können entfernte Systeme komfortabel vermessen werden

Download und Installation

- Aktuelle Version:
  `http://www.benchit.org/downloads/files/benchit-release.tar.gz`
- Herunterladen und Entpacken und in das Verzeichnis wechseln
- Starten der BenchIT GUI via *./GUI.sh*

Einstellungen

Zuerst müssen einige initiale Angaben zum System gemacht werden
(Architektur Kürzel, Prozessor Geschwindigkeit)
Der "mouse-over" Info Dialog enthält eine genauere Beschreibung

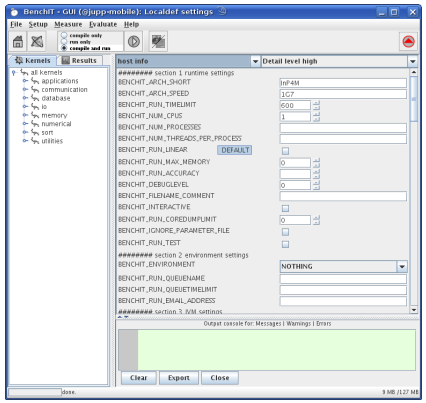


Abbildung: LOCALDEF-Editor

Einstellung

Einige der Werte werden vom System automatisch ermittelt - diese können so verbleiben.

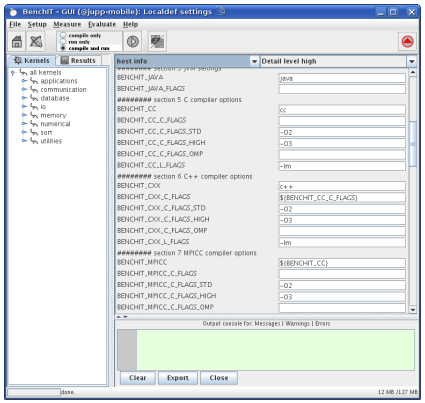
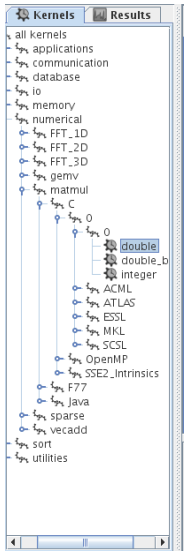


Abbildung: Automatisch gesetzte Werte für die Compiler-Einstellungen

Kernel Verzeichnisbaum



In der Standardansicht befindet sich links der sogenannte *kernel tree*. Dies ist eine strukturierte Liste aller *Messkerne*, also der Algorithmen / Programme welche zur Leistungsmessung verwendet werden. Bitte wählen Sie hier numerical→matmul→C→ 0 → 0 →double.

Struktur des *Kernel Tree*

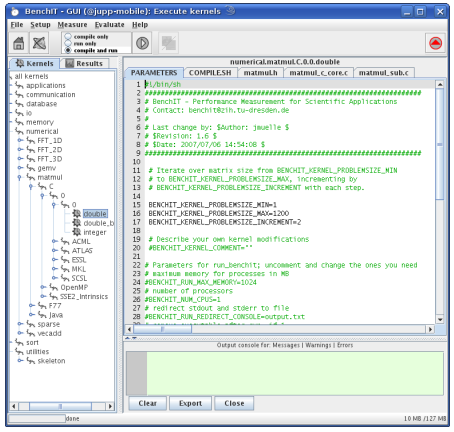
Die Struktur des *Kernel Tree* entspricht der Struktur des *kernel* Verzeichnisses in der *BenchIT* Distribution. Im *_reference_* Ordner befinden sich alle Kerne welche nur einen C-Compiler voraussetzen und sich daher für einen ersten Überblick ohne großen Konfigurationsaufwand eignen. Die Hierarchie der Ordner folgt folgendem Prinzip:

$$\underbrace{\text{numerical}}_{\text{category}} / \underbrace{\text{matmul}}_{\text{algorithm}} / \underbrace{\text{C}}_{\text{language}} / \underbrace{0}_{\text{parallel library}} / \underbrace{0}_{\text{misc library}} / \underbrace{\text{double}}_{\text{data type}}$$

Demnach haben sie soeben eine nicht-parallele, von keiner Bibliothek abhängige C-Version des Matrix Multiplikations Algorithmus ausgewählt.

Die erste Messung: Parameters

Sobald Sie den *matmul* Kern im *Kernel Tree* auswählen, verändert sich das Hauptfenster zu einem Editor. Die verschiedenen Dateien eines Kerns stehen dort über Tabs zur Verfügung. Im Standardfall wird die Datei *PARAMETERS* geöffnet welche die Parameter des Kerns für den Messlauf bestimmt.



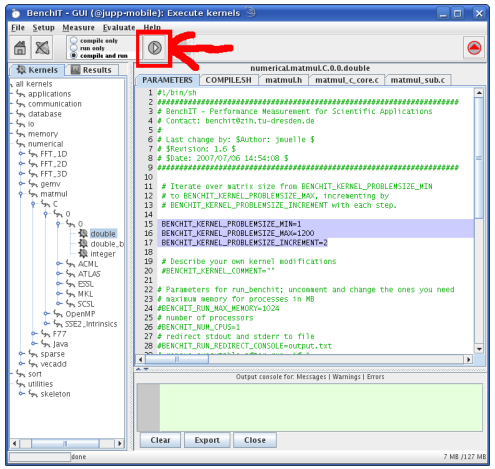
Die erste Messung: Parameters

Mit den Standardeinstellungen des Matrix Multiplikation Kerns iteriert der Algorithmus über die Matrixgröße von 1 bis 1200. Zu Testzwecken sollten die Werte wie folgt abgeändert werden:

```
15  BENCHIT_KERNEL_PROBLEMSIZE_MIN=1
16  BENCHIT_KERNEL_PROBLEMSIZE_MAX=400
17  BENCHIT_KERNEL_PROBLEMSIZE_INCREMENT=7
```

Die erste Messung: Start!

Damit sollten alle Vorbereitungen für die erste Messung abgeschlossen sein. Über den *Compile and Run* Button in der Tool-Bar kann nun die Messung gestartet werden.



Die erste Messung: Überwachen des Messlaufs

Im unteren Abschnitt der GUI kann der Ablauf der Messung verfolgt werden.

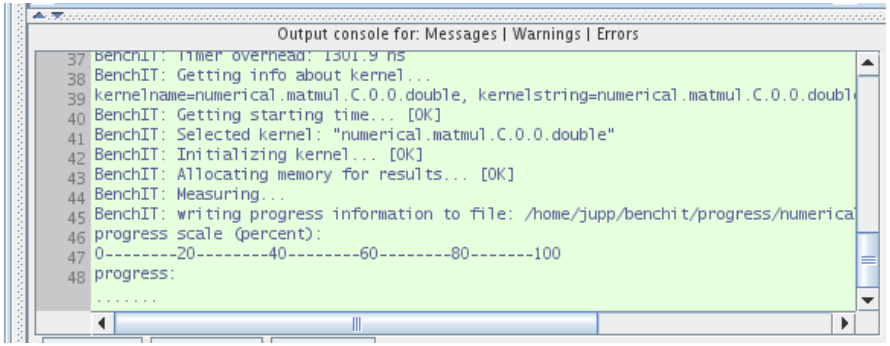
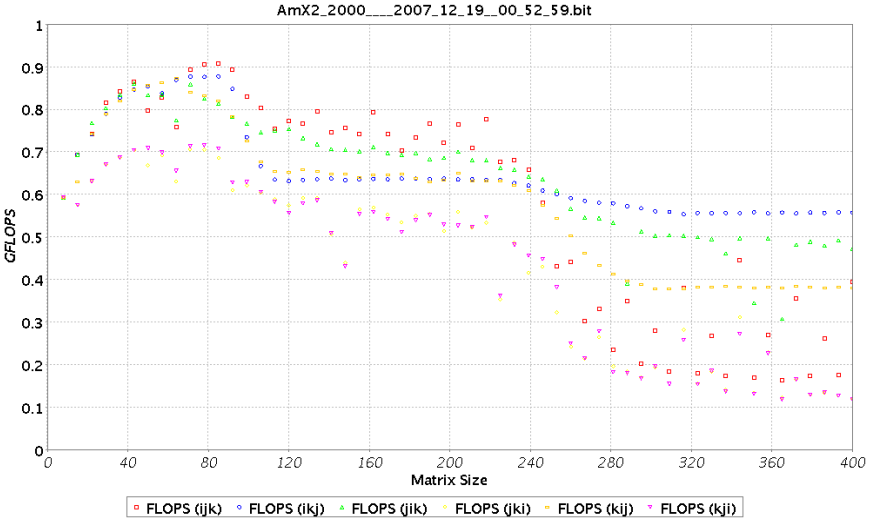


Abbildung: Ausgabe der Kommandozeile

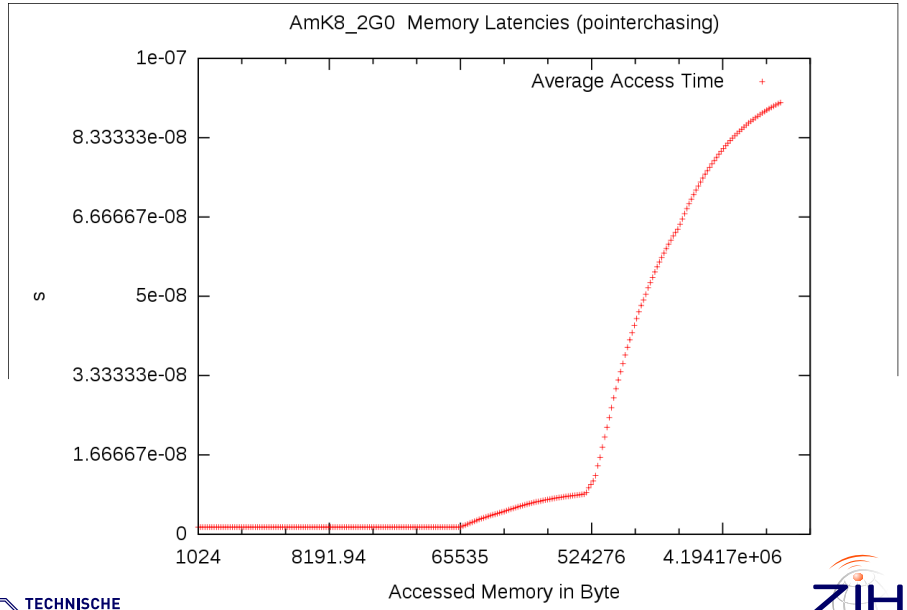
Die erste Messung: Visualisierung der Ergebnisse

- Sobald die Messung abgeschlossen ist, kann das Ergebnis über den '*Results*'-Tab in der rechten oberen Ecke über dem *Kernel Tree* visualisiert werden
- Die Strukur des *Result Tree* entspricht der des *Kernel Tree* (es werden nur Kerne mit Ergebnissen angezeigt)
- Dort sollte sich jetzt ein Knoten 'numerical' befinden (und sog. *Mixer* - dazu später mehr)
- Folgen Sie dem *Result Tree* bis zur Ergebnis-Datei (*Result File*)
- Mit einem Klick auf diese Datei wird der zugehörige Graph gezeichnet (*Result Plot*)

Die erste Messung: Beispiel-Plot 1



Die erste Messung: Beispiel-Plot 2



Die erste Messung: Auswertung

Als Übung und zur Veranschaulichung sollen folgende 2 Fragen dienen:

- Welche Permutation der Matrix Multiplikation ist optimal für Ihren Rechner (Plot 1)?
- Schätzen Sie die Größe des L1/L2-Cache (Plot 2)?

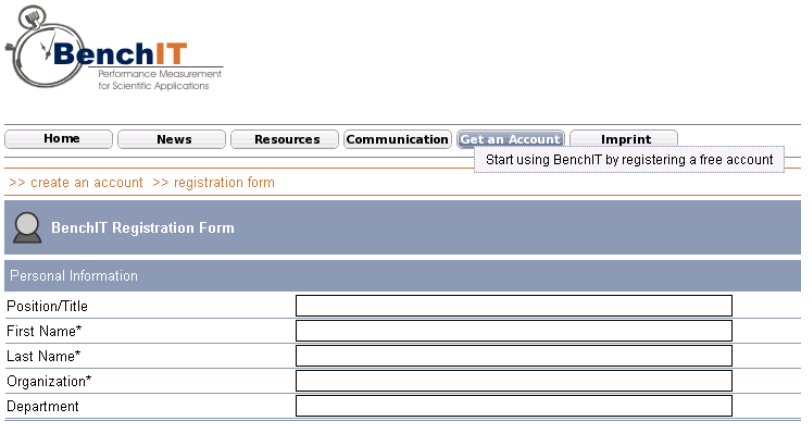
Enfernte Messungen

Nun wird der selbe Kern (mit anderen Paramtern) auf dem Cluster "deimos" ausgeführt.

- Legen sie mit Hilfe der GUI einen Ordner auf deimos an (Setup→Create a remote folder).
- IP: `deimos.hrsk.tu-dresden.de`, Nutzer: `username`, Ordner: `benchit`.
- Es sind folgende Paramter zu verwenden (e.g. min=1, max=1400, increment=1).
- Laden Sie die LOCALDEFS für deimos (Setup→Localdefs, File→Load→deimos).
- Passen Sie die Anzahl der CPU's und das Debug-Level an (BENCHIT_NUM_CPUS=1, BENCHIT_DEBUGLEVEL=0).
- Nun kann der Kern auf deimos ausgeführt werden (Measure→Execute in remote folder).

Ergebnisse vergleichen und hochladen

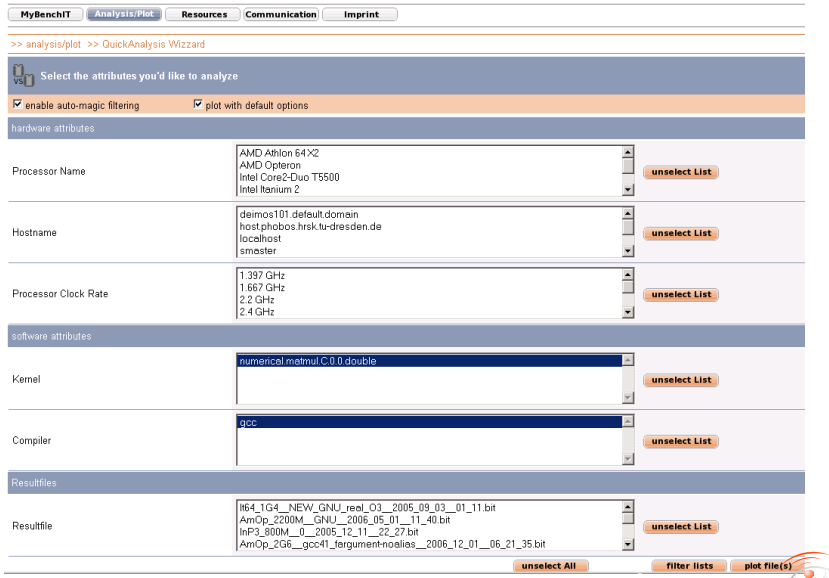
- Legen Sie auf der BenchIT-Website einen Account an (Button: Get an Account).
- Mit der Aktivierungsmail kann nun der Account frei geschaltet werden.



Ergebnisse vergleichen und hochladen

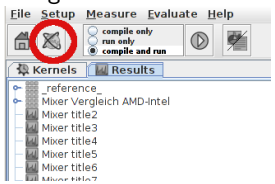
- Um Ergebnisse auf die Seite laden zu können müssen die LOCALDEFS stimmen und zu einem gewissen Grad ausgefüllt sein. Falls dies bei Ihren Ergebnissen nicht der Fall ist können sie trotzdem die Ergebnisse auf der Seite für den Vergleich verwenden.
- Benutzen Sie auf der Webseite die Menüeinträge 'Analysis/Plot' und 'QuickAnalysis Wizzard'.
- Wählen Sie den Kern (numerical.matmul.C.0.0.double) und den Compiler (gcc).
- Nun sollte eine Ihrem Rechner ensprechende Maschine gewählt werden - danach klicken Sie auf 'plot file(s)'.

Ergebnisse vergleichen und hochladen

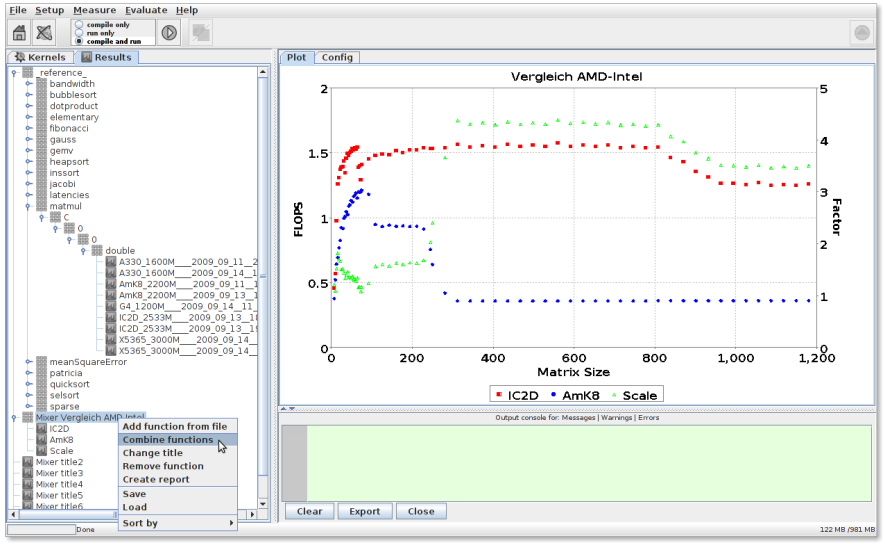


Mixer und Online-Mixer

- Sie können per "drag&drop" verschiedene Messungen in einen Mixer ziehen
- Falls ein Plot mehrere Funktionen enthält können Sie wählen welche in den Mixer übernommen werden
- Im Mixer können so verschiedene Messläufe gleichzeitig & gemeinsam dargestellt werden
- Außerdem können einfache arithmetische Operationen auf die Messreihen angewendet werden
- Zusätzlich gibt es einen *Online-Mixer*, dieser erlaubt mit einem gültigen Login den Zugriff auf die Datenbank mit den Messreihen der Webseite
  - Der Button links oben unter der Menüleiste startet den Loginvorgang
  - Nach dem Login wird der OnlineMixer unter den anderen Mixern geListet:



Mixer



Eigene Kerne schreiben

- Zu diesem Zweck gibt es sogenannte "skeletons" für alle unterstützten Sprachen
- Eine schrittweise Anleitung findet sich unter folgendem Link: `http://www.benchit.org/wiki/index.php/How_to_write_a_Kernel`

High Performance Computing Linpack Benchmark (HPL)

- Gleichungslöser für dichtbesetzte Matrizen
- Mit Zufallswerten gefüllt
- Doppelte Genauigkeit (64bit)
- Verteiltes Speichermodell
- Benötigt MPI (1.1)
- Benötigt Basic Linear Algebra Subprograms (BLAS)
- Oder Vector Signal Image Prozessing Library (VSIPL)

High Performance Computing Linpack Benchmark (HPL)

- Link: `http://www.netlib.org/benchmark/hpl/hpl-2.0.tar.gz`
- README und INSTALL lesen
- Initialen Lauf ausführen
- TUNING lesen
- Versuchen Sie den Initialen Lauf durch Anwendung der Tuning-Tipps zu verbessern
- Für die BLAS gibt es verschiedene Implementationen:
  - Referenzimplementation von netlib: `http://www.netlib.org/blas/`
  - Autotuning mit ATLAS: `http://math-atlas.sourceforge.net/`
  - Intel optimierte MKL: http://software.intel.com/en-us/intel-mkl/