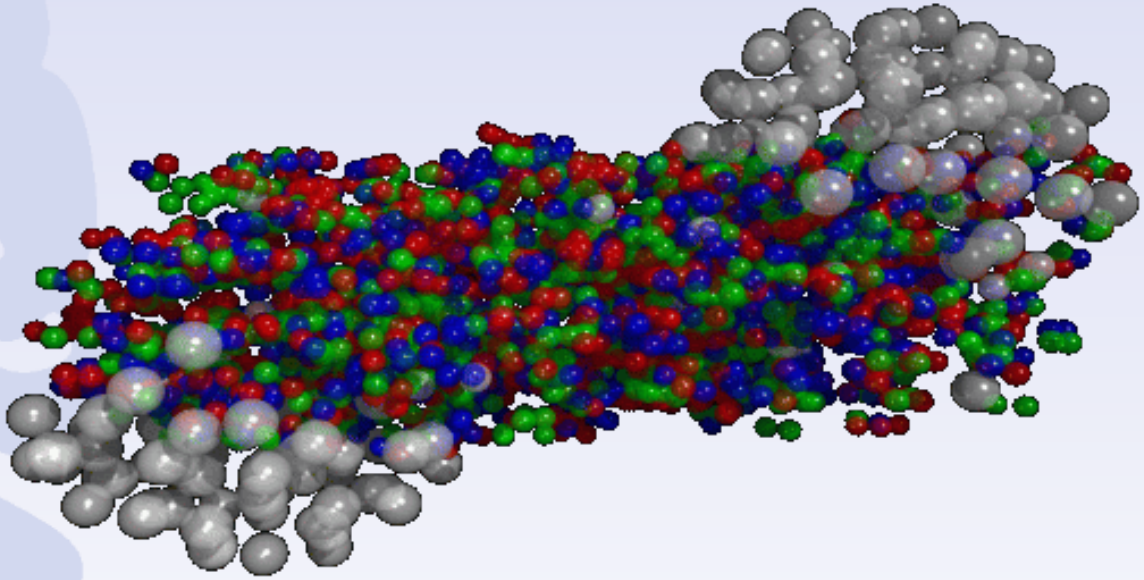
GOETHE UNIVERSITÄT FRANKFURT AM MAIN

Refactoring the UrQMD Model for Many-Core Architectures



Mathias Radtke

Semiar: Softwaretechnologie (WS 2013/2014

Goethe-Universität Frankfurt

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

Agenda:

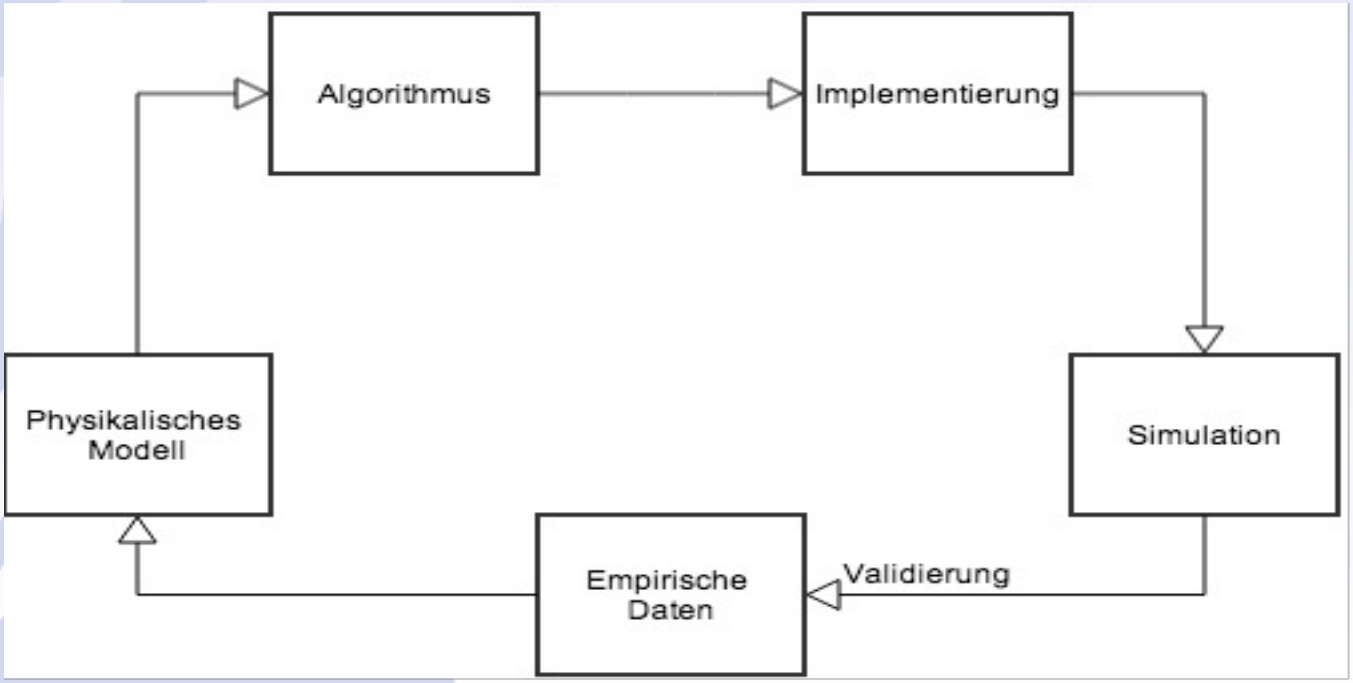
1. UrQMD
2. CPU Vs. GPU
3. Von FORTRAN zu C++/OpenCL
4. Ergebnisse
5. Schlusswort

1. UrQMD

- Abkürzung für Ultra relativistic Quantum Molecular Dynamics
- Physikalisches Modell zur Beschreibung des Transports, der Kollision, der Streuung und des Verfalls von Kernteilchen
- Vor fast 20 Jahren implementiert in FORTRAN 77 und immer weder erweitert
- Codedesign und Effizienz von geringem Interesse
- Großteil der Berechnungen verwendet für relativistisch hydrodynamischen Verhalten von Kernteilchen (SHASTA)
- SHASTA ist rein sequentiell
- UrQMD ist eine große Sammlung von physikalischen Formeln in implantierter Form

1. UrQMD

- Neue Erkenntnisse werden mit UrQMD gewonnen und implementiert
- Ergebnisse aus Experimenten werden in UrQMD hinzugefügt
- Ein stetiger Kreislauf entsteht



1. UrQMD

- Trotz des Alters ein wertvolles Programm für die Physik
- Keine neue Softwaretechnologie und kaum neue Hardwaretechnologie seit Implementierung

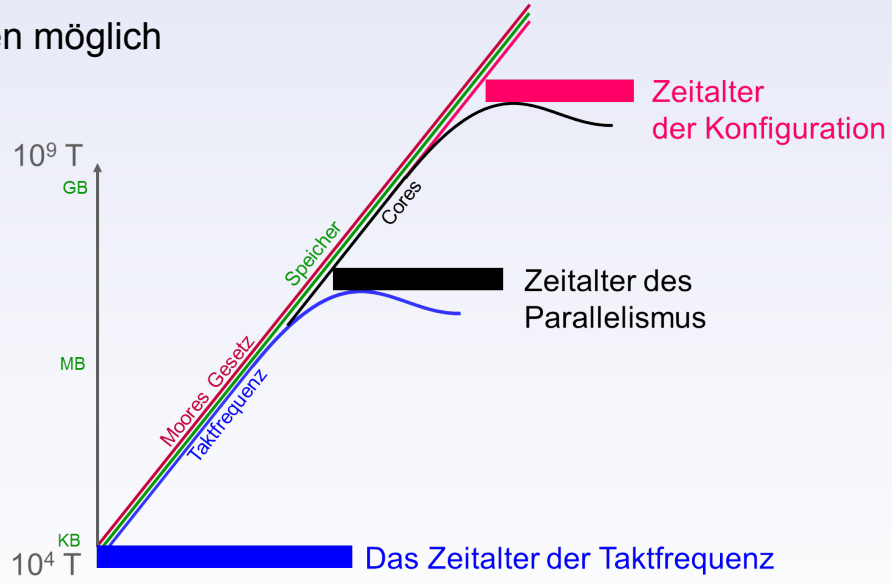
FORTRAN:

- Problemorientiert
- Keine globalen Variablen, **common blocks** als Ersatz
- goto-Anweisungen, der weltberühmte Spaghetti-Code
- Kaum Unterstützung von Parallelität
- Es gibt Erweiterungen um FORTAN auf HPCs zu benutzen
    - Nicht Sinn der Sache
- Refactoringtool **f2c** empfehlenswert, nur simpelste Anweisungen
- Experte leistet bessere Arbeit

1. UrQMD

ZIEL:

- UrQMD an neue Soft- und Hardwaretechnologie anpassen
    - C++/OpenCL auf CPUs und GPUs
- UrQMD muss gleiche Ergebnisse liefern wie unter FORTRAN
- Verbesserte Leistung kaum noch über Taktraten möglich



GOETHE UNIVERSITÄT FRANKFURT AM MAIN

2. CPU Vs. GPU (Ochsen gegen Hunde)

- CPUs haben ausgeklügelte Parallelität
  - Branch-Prediction
  - Pipelining
  - Multithreads
  - Out-of-Order-Execution
- GPUs sind Arbeitsmaschinen
  - Mehr als 1000 Prozessoren (AMD Radeon HD 7970, 2048 Streamprozessoren)
  - Takt von ca. 1GHz
  - 4 Threads pro Streamprozessor, 250MHz / Thread
  - Ideal für Fließkommaberechnung

2. CPU Vs. GPU (Ochsen gegen Hunde)

- Speicheranbindung
  - DDR3-2133, 17,066GByte/s (peak) [1]
    - 128GB im Desktop Rechner möglich, 256GB im Server-Blade
  - GDDR5, 576 Gbyte/s (peak), AMD Radeon 7990 [2]
    - 6GB, keine Erweiterung möglich
    - Daten müssen entweder in/aus den/m Hauptspeicher oder Daten müssen verkleinert werden

[1] http://www.elektronik-kompendium.de/sites/com/1312291.htm
[2] http://www.pc-erfahrung.de/fileadmin/php-skripte/grafikrangliste_info.php?ID=355&ARCH=0

2. CPU Vs. GPU

Ziel:

- Symbiose zwischen CPU und GPU
- Vorteile beider Architekturen miteinander arbeiten lassen
- OpenCL bietet diese Möglichkeit
    - Erweiterbar auf verwendete CPU/GPU
    - Nachteil: Portabilität leidet
- NVIDIA CUDA verwendet nur NVIDIA Hardware, OpenCL hingegen alle Hersteller

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

3. Fortran zu C++/OpenCL

- SHASTA am zeitaufwendigsten
- Ideal für das Refactoring
- Realisierung in zwei Schritten
    1. Ein Hybrid aus FORTRAN und C++ wird erstellt.
        - FORTRAN Routinen und C++-Funktionen
    2. Vollständige C++ Version mit allen Eigenschaften der FORTRAN-Version
- C++-Version wird vollständig in OpenCL überführt

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

www.uni-frankfurt.de

3. Fortran zu C++/OpenCL

- Stolpersteine sind vorhanden
    - Keine Klassen auf GPUs
    - Kernel-Orchestrierung, Bufferablauf und Queue in einer Klasse
    - Einige Klassen aus SHASTA ausgelassen, der Algorithmus arbeitet dennoch wie er soll (changegrid)

- Der Hybrid:
    - Mischung aus FORTRAN-Routinen und C++-Funktionen
    - Test für numerische Gleichungen
    - Verhalten von SHASTA nachgeahmt

3. Fortran zu C++/OpenCL

- Die C++-Version kann verschiedene Parameter zum Start als Eingabe annehmen
  - Zeit, Eingabedateien und Partikelzustand
- Konstruktor in C++ kümmert sich um die Eingabeparameter
- FORTRAN-Version greift mehrmals auf Massenspeicher zu und beinhaltet Verzweigungen
  - Verlängerung der Laufzeit

3. Fortran zu C++/OpenCL

```
else if (eos.eq.4) then
        call readeos.3()
else if (eos.eq.5) then
        call readeos.3()
end if
...
if (eos.eq.1) then
        ...
        else if (eos.eq.3) then
        ...
        else if ((eos.eq.4).or.(eos.eq.5)
                                .or.(eos.eq.2)) then
                if (e.lt.400.0d0) then
                        if ((e.lt.0.1d0).and.
                        (n.lt.0.02d01)) then
```

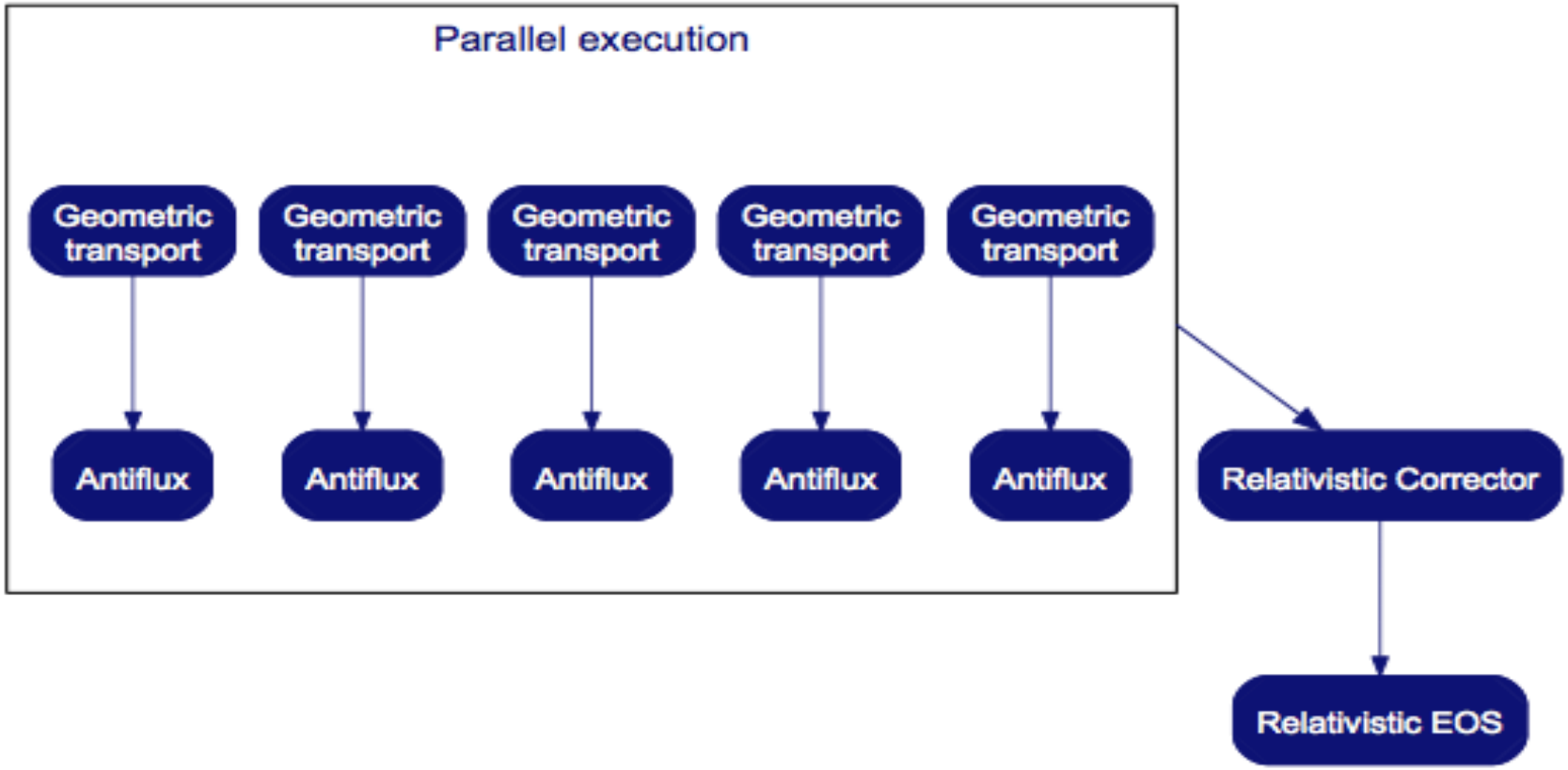
3. Fortran zu C++/OpenCL

```
EqofState = EOS;
switch(Data->eos){
        case 1: EOS = new EqofState1(); break;
        case 2: EOS = new EqofState2(); break;
        case 3: EOS = new EqofState3(); break;
        case 4: EOS = new EqofState4(); break;
        case 5: EOS = new EqofState5(); break;
        deafult: EOS = new EqofState1(); break;
}
```

3. Fortran zu C++/OpenCL

- Die Erstellung des Hybriden wurde stetig Kontrolliert
- Jede neue Methode wurde direkt auf Funktionalität geprüft, sodass kein falsches Ergebnis abhängige Berechnungen verfälscht.
- Tests und Refactoring von Hand durchgeführt
- C++ und FORTRAN behandeln mathematische Gleichungen nicht gleich, Ergebnisse unterscheiden sich leicht
- C-SHASTA (C++/OpenCL-Version) Berechnet tausende von Datenströmen, ein guter Datenfluss ist notwendig
    - So viele Daten wie möglich parallel Berechnen bevor ein sequentieller Teil kommt

3. Fortran zu C++/OpenCL



3. Fortran zu C++/OpenCL

- Grafische Ausgabe über ein 3D-Gitter mit 8Mio Zellen
  - Im Worst-Case 8Mio Informationen Berechnen und grafisch ausgeben
- GPUs haben weniger Speicher als die CPU
  - Verringerung der zu speichernden Informationen
  - Wir rechnen in einem 1D-Array
    - Viel mehr Berechnungen notwendig, aber:
    - Geringerer Speicherbedarf
    - Ergebnisse schneller vorhanden
- C-SHASTA ist erweiterbar mittels Modulen
  - Keine Änderung im Code nötig

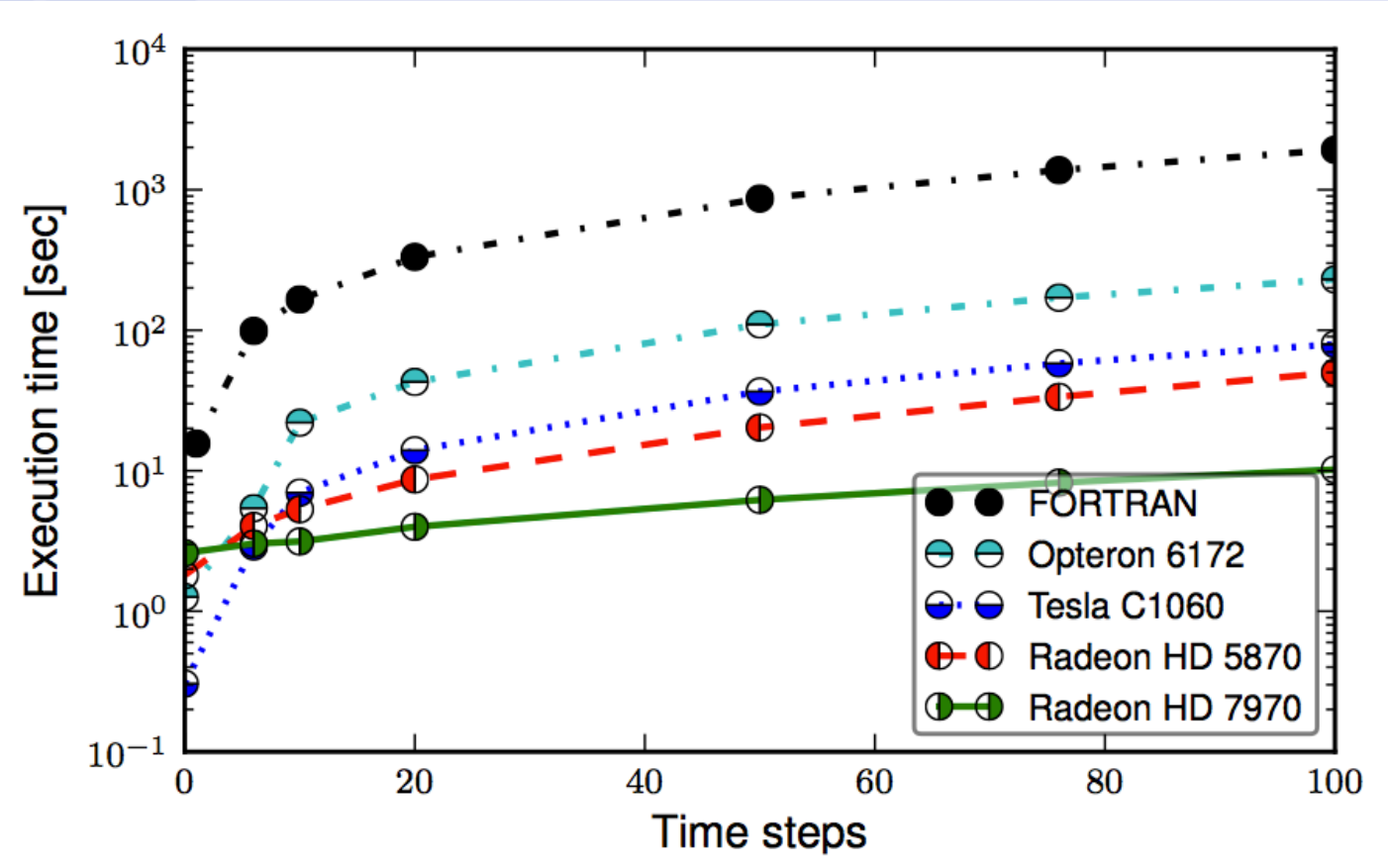
GOETHE UNIVERSITÄT FRANKFURT AM MAIN

4. Ergebnisse

- C-SHASTA wurde auf folgender Hardware getestet
  - NVIDIA Tesla C1060
  - AMD Radeon HD 5870
  - AMD Radeon HD 7970
  - AMD Opteron 6172 (24 Cores)
- Die Meisten Berechnungen wurde auf dem LOEWE-CSC gerechnet
- Ergebnisse wurden mit realistischen Daten gerechnet
- Tests:
  - Au-Au Kollision (Goldpartikel)
  - Eine expandierende Kugel mit r=2fm (1fm = 10^{-15}m)

4. Ergebnisse

- Au-Au-Kollision

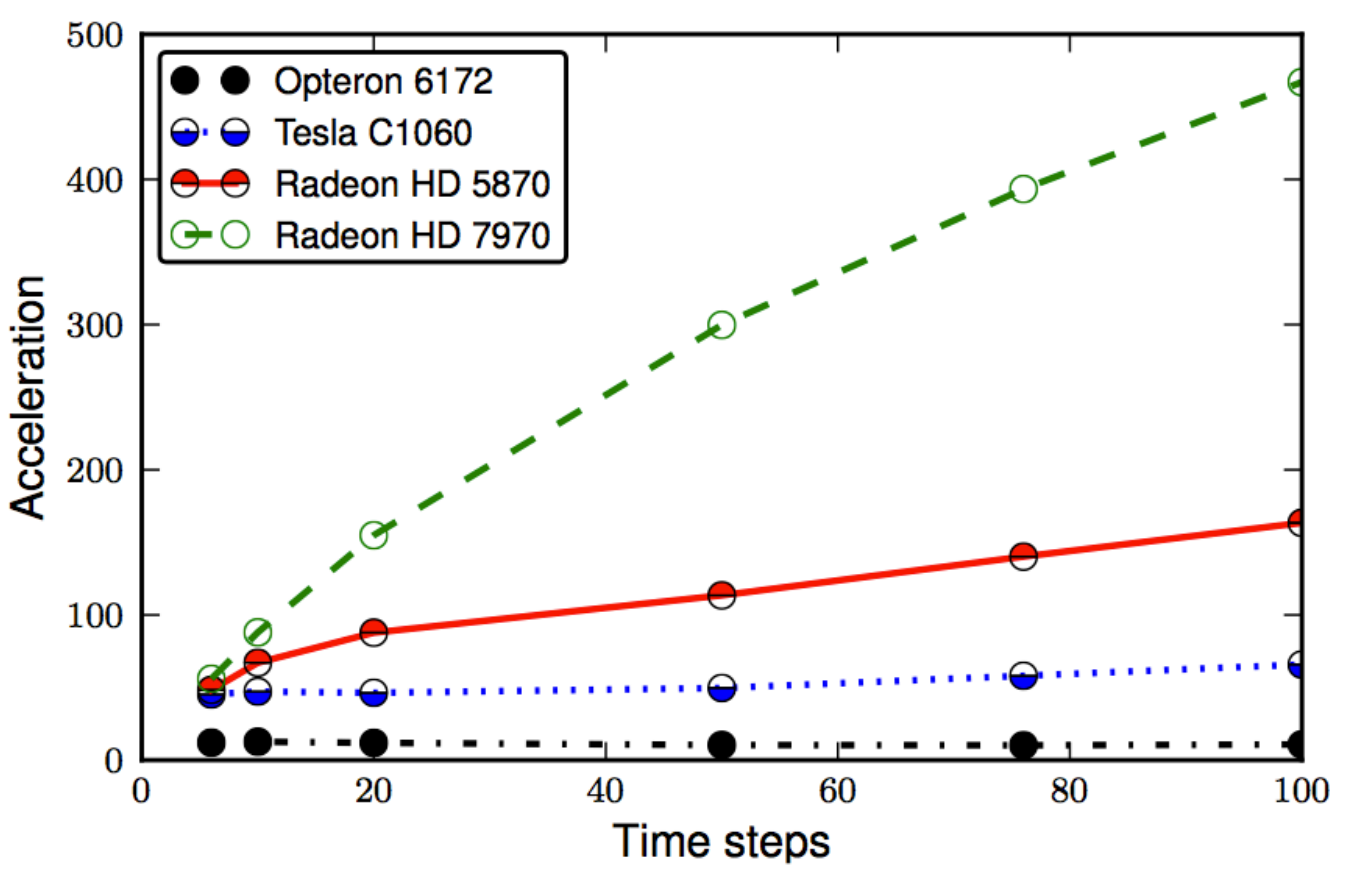


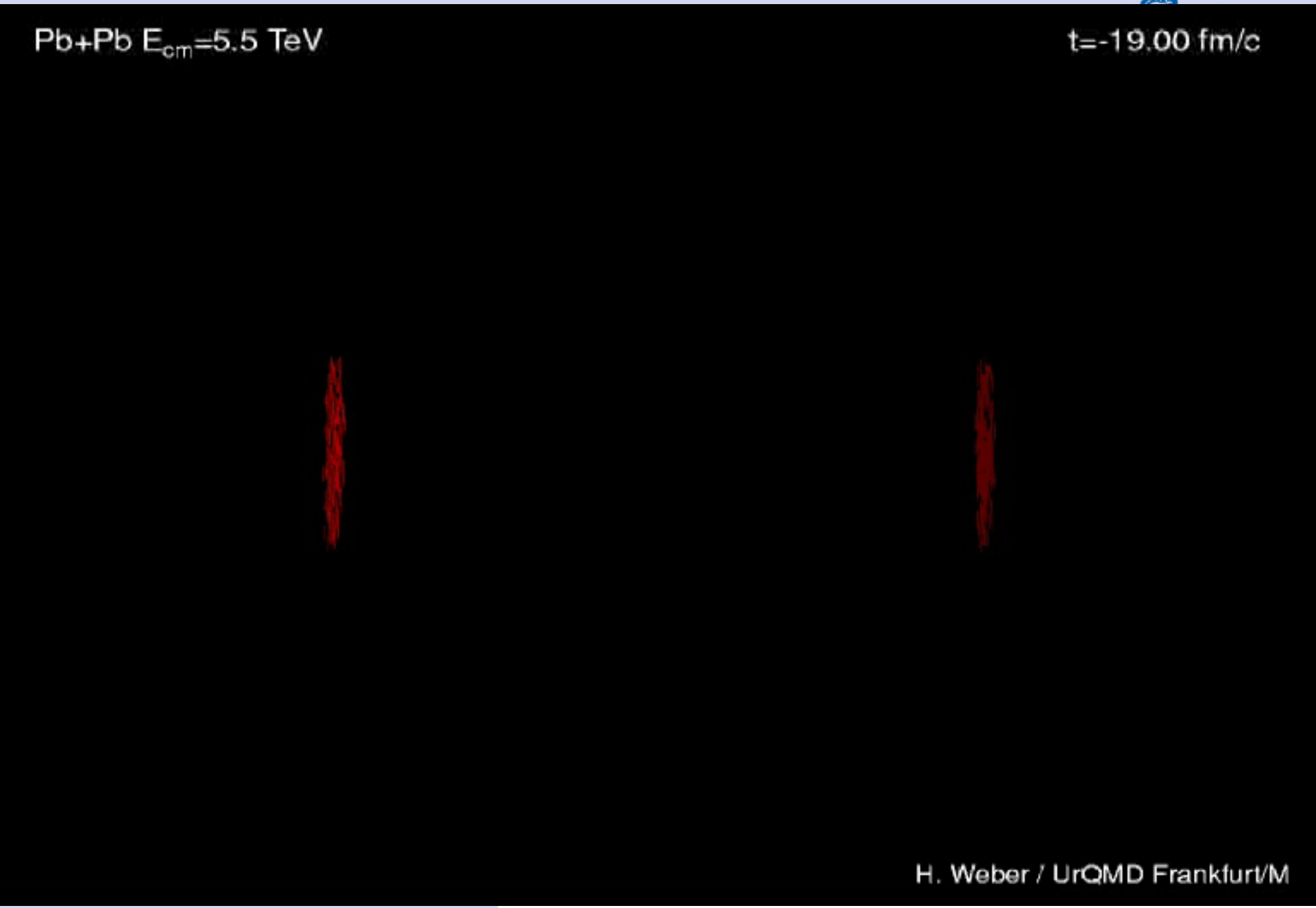
4. Ergebnisse

- Au-Au-Kollision
  - Laufzeit:
    - FORTRAN: 75Min
    - AMD Radeon HD 5870: 30 Sekunden (Faktor 160)
    - AMD Radeon HD 7970: weniger als 10 Sekunden (Faktor 460)

4. Ergebnisse

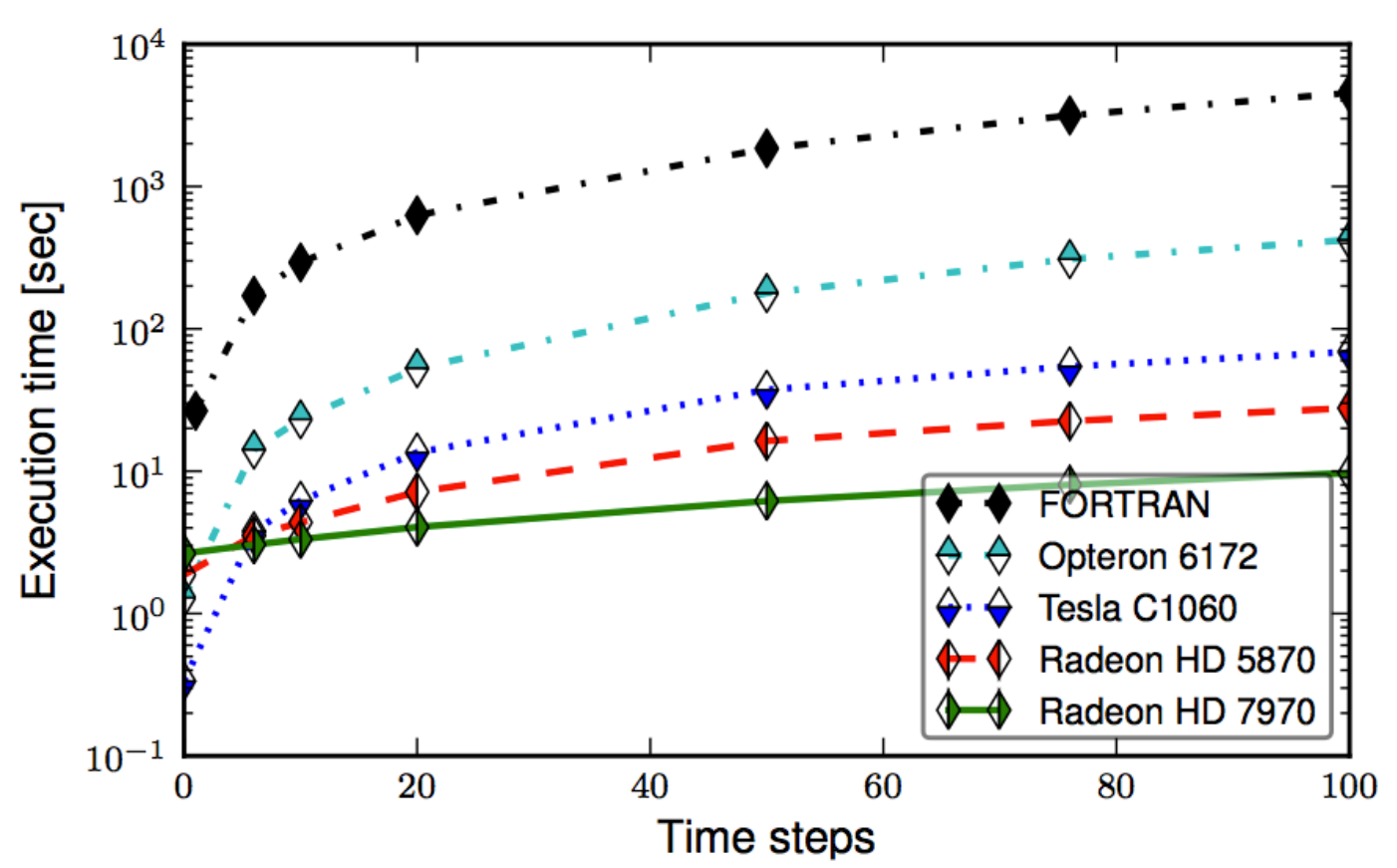
- Au-Au-Kollision




Pb+Pb E_{cm}=5.5 TeV
t=-19.00 fm/c
H. Weber / UrQMD Frankfurt/M

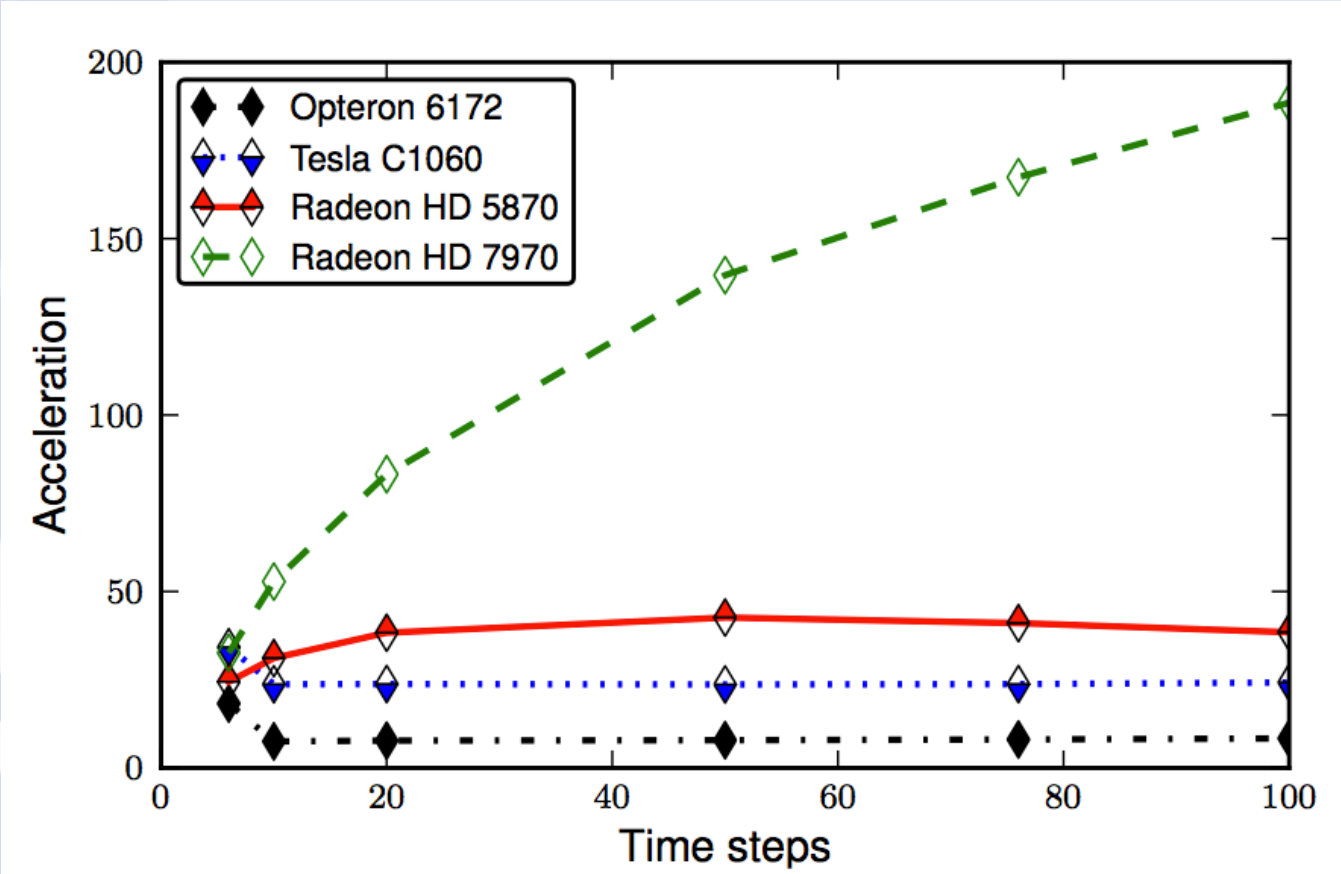
4. Ergebnisse

- Expandierende Kugel mit r = 2fm



4. Ergebnisse

- Expandierende Kugel mit r = 2fm



GOETHE UNIVERSITÄT FRANKFURT AM MAIN

www.uni-frankfurt.de

5. Schlusswort

- Refactoring im Rahmen einer Dissertation angefertigt
    - Nur ein Teil von UrQMD, benötigte aber die meiste Rechenzeit
- Enorm gesteigerte Laufzeit, Faktor 460 im besten Fall