Implementation des MAFIA E–Moduls auf APE–100 Rechnern

F. Neugebauer (fneug@ifh.de), DESY Zeuthen

11. Mai 1999

- Motivation
- Problemstellung
- Tao-Code
- Ergebnisse
- Ausblick

Motivation

- MAFIA: exzellentes Werkzeug zum Berechnen elektromagnetischer Felder (lang erprobt, Tk–interface)

- Allerdings: Problemgröße durch Hauptspeicher und Rechenkapazität begrenzt, komplizierte Probleme meist nur näherungsweise behandelt (rotationssymmetrisch)
  Beispiel: TESLA (bislang keine 3–dimensionale numerische Simulation)

- Portierung einzelner Teile von MAFIA auf APE–100
  Großer Hauptspeicher $\rightarrow$ viele Gitterpunkte
  Große Rechenkapazität $\rightarrow$ Rechenzeitgewinn

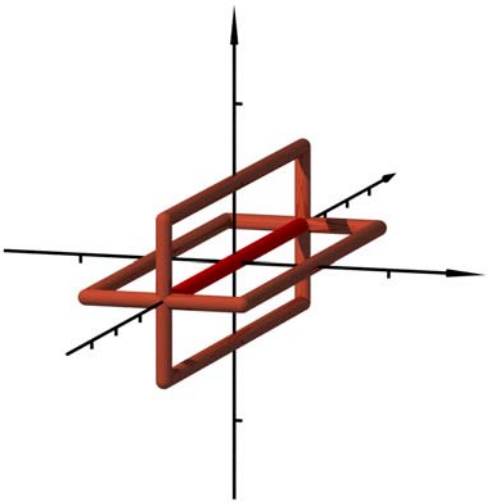
- Aufgabe: Verbindung der Vorzüge von MAFIA mit der Rechenleistung einer APE–100
  MAFIA nicht neu programmieren, sondern Material/Geometrie–eingabe und Visualisierung der Ergebnisse durch MAFIA nutzen

- Erste Versuche mit dem E–Modul, Bestimmung der n tiefsten Eigenmoden in supraleitenden Hohlraumresonatoren, konsequent drei–dimensional

- MAXQ: Maxwell Gleichungen auf Quadrics–Rechnern, Projekt bei DESY

Problemstellung

- Maxwellsche Gleichungen mittels FIT (Finite Integration Technique)[1] in algebraisches Gleichungssystem transformiert
- Kontinuitätsgleichung und $divB = 0$ bleiben exakt gültig (kein Diskretisierungsfehler an dieser Stelle), Freiheitsgrade des Systems sind die Weg– bzw. Flächenintegrale der Feldgrößen auf dem Gitter
- Wellengleichung transformiert sich zu einem linearen Eigenwertproblem $A\vec{x} = \lambda\vec{x}$
- Matrix–Vektor–Multiplikation $A\vec{x}$ entscheidend für die Effizienz der Rechnung



Nur “benachbarte” Gitterspannungen werden über die Systemmatrix miteinander verknüpft
$\longrightarrow$ ideal geeignet für APE–100

[1] T. Weiland, AEÜ, Vol. 31, pp. 116–120 (1977)

TAO I

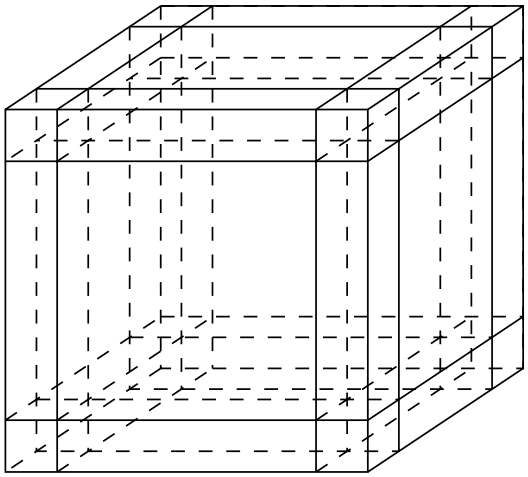
- Für APE relevant: Wie sieht die Matrix–Vektor–Multiplikation aus?
  Ausschnitt aus MAFIA (orginal code):

```
      do 300 ip = nuv+1,np
        f2(ip)=-shift*f1(ip)-rarray(kkmdb1+ip-1)*
     1       ( (rarray(kkbda3    +ip-1)-rarray(kkbda3-nu+ip-1))
     2        +(rarray(kkbda2-nuv+ip-1)-rarray(kkbda2   +ip-1)))
 300  continue

      do 310 ip = nu+1,nuv
        f2(ip)=-shift*f1(ip)-rarray(kkmdb1+ip-1)*
     1       ( (rarray(kkbda3    +ip-1)-rarray(kkbda3-nu+ip-1))
     2        +(                       -rarray(kkbda2   +ip-1)))
 310  continue

      do 320 ip = 1,nu
        f2(ip)=-shift*f1(ip)-rarray(kkmdb1+ip-1)*
     1       ( (rarray(kkbda3    +ip-1)                        )
     2        +(                       -rarray(kkbda2   +ip-1)))
 320  continue
```

  f1 ist der input–Vektor, f2 das Ergebnis der Multiplikation. Die Systemmatrix ist überwiegend implizit programmiert, der Index ip durchläuft alle Gitterpunkte ($ip = 1, np$). Die vollständige Multiplikationsroutine besteht aus 13 Loops des obigen Typs.

- Nachbarschaftsrelationen durch konstante offsets
  (z.B. $-nu =$ linker Nachbar in v–Richtung, etc.)
- Problem: keine periodischen Randbedingungen,
  Aufspaltung in kleinere loops in MAFIA $\rightarrow$ WHERE/ELSEWHERE in TAO

TAO II



Aufteilung des Hauptspeichers auf jedem Knoten der APE–100 in 27 Segmente.

- Segmentierung des Hauptspeichers um Sonderfälle am Rand des Rechengebietes zur Compile–Zeit abzufragen.
- In jedem Segment sind spezielle Nachbarschaftsoffsets “gültig”
  (über include-files definiert)
- Nur auf den Randsegmenten ist WHERE/ELSEWHERE notwendig!
- Alternative Programmiervariante: Frames, kosten extra Speicher, kein Geschwindigkeitsgewinn

TAO III

- Beispiel aus TAO:

```
 ipl = 1 + nuv

 do iw = 1,nw-2
 do iu = 1,nu-2

  iv = 0
/include "standard_offsets.hzt"
/include "v_lower_border_offsets.hzt"

  i  = ipl

t_w[i] = dc3[i] * ( ...
         s_v[i] - s_v[i + u_p] ...
       + s_u[i + v_p] - s_u[i] )

 enddo
 enddo
```

- Koordination der 27 Segmente mit den 13 loops der Matrix–Vektor–Multiplikation mittels sed/shell–script.
  Änderungen am code werden an genau einem File genau einmal vorgenommen, danach wird der eigentliche TAO–code automatisch generiert.
  (enorme Zeitersparnis und Tippfehlerreduktion beim Programmieren)

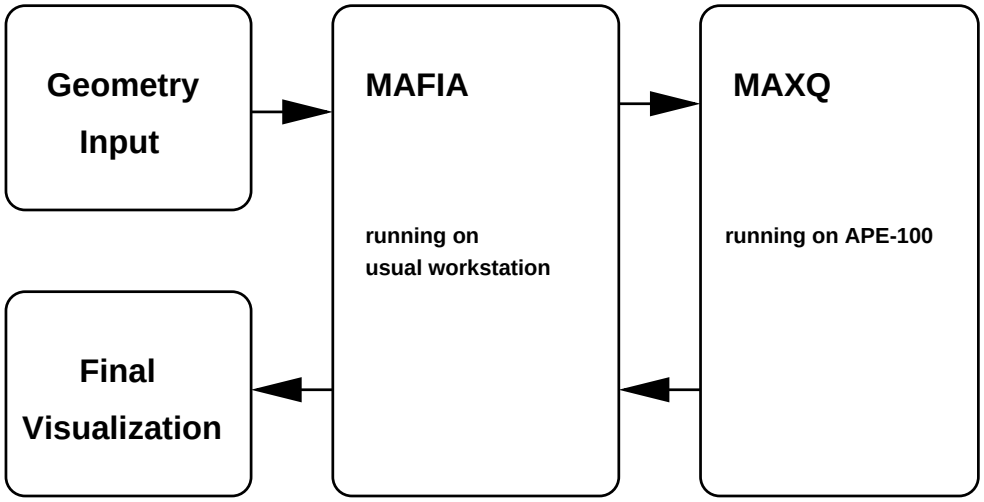
TAO IV

- Skalarprodukte, Multiplikation mittels “extract/replace” optimiert.

- Performance der Multiplikation bei 15% (hängt von der Burstlänge und Anzahl der Gitterpunkte ab)

- IO mittels “binary multidata” $\rightarrow$ 500kB/s (Lesen/Schreiben kann länger als die Rechnung dauern)

- Interface zu MAFIA mittels MTK (MAFIA toolkit), Umordnung der Matrix (linear / verteilt auf die APE–Knoten) notwendig



Ergebnisse

- Speicherbedarf beim Conjugate–gradient Eigenwertsolver[2] (QH2b):

Anzahl Eigenvektoren	maximale Anzahl Gitterpunkte
1	60.000.000
2	51.000.000
5	36.000.000
10	24.000.000
100	3.400.000

- Rechenzeitvergleich: mittleres Problem 80x80x80 (500 000 Gitterpunkte)

  $\longrightarrow$ 96 Minuten auf paris, 5 Minuten auf QH4 : Faktor 18
  Rechenzeit skaliert linear mit der Anzahl der Gitterpunkte.

- Problem: Wartezeiten in der queue um 2 Größenordnungen länger als die eigentliche Rechnung $\longrightarrow$ Problemgröße steigern: MAFIA Experten gefragt!

- Nebenbedingung: Anzahl der Gitterpunkte in xyz–Richtung muß durch die Anzahl der Prozessoren in xyz–Richtung teilbar sein.
  Scheint nicht trivial zu sein (automesh=no)

[2] Th. Kalkreuther, and H. Simma, 'An accelerated conjugate gradient algorithm to compute low-lying eigenvalues – a study for the Dirac operator in SU(2) lattice QCD', DESY 95-137, HUB–IEP–95/10 (1995)

Zusammenfassung und Ausblick

- E–Modul von MAFIA erfolgreich auf APE–100 portiert
- Rechnungen für kleinere Gitter zeigen Übereinstimmung von MAFIA und MAXQ
- Aufruf an die MAFIA Nutzer:
  realistische Geometrien entwerfen für 10^6–10^7 Gitterpunkte
- MAFIA Nutzer: Bedarf beim T–Modul (d.h. Zeitbereichsrechnungen, Differentialgleichung 1. Ordnung) $\rightarrow$ ebenfalls portieren