Effiziente Programme WS2010/11

Gruppe 105

Bernhard Kragl
Michael Schöndorfer
Benjamin Maurer

Ist-Zustand

- Testaufruf
  - `gcc -O spXX.c`
  - `papiex -e PAPI_TOT_CYC -e PAPI_TOT_INS -e PAPI_BR_MSP a.out < input-bench-littleendian >/dev/null`
- Ergebnis:

Cycles:	1.61048e+08
Instructions:	2.43945e+08
Branch mispredictions:	1.71534e+06

Erste Überlegungen

- putchar – Optimierung durch Ausgabe-Buffer
- Verbesserung in Testprogramm
- Nicht implementiert →
  Bei Ausgabe nach /dev/null egal

Bereits gezeigt

- Schleifen herunter zählen (Prozessor NULL Flag)
- Mehrfache Berechnung eliminieren

Macro Substitution

- Var maxstates → macro MAX_STATE

```
3918   for (k=MAX_STATE-1; k>=0; k--) {
3919     trans[k] = inst[k];
3920     trans[k].no_transition = 1;
3921   }
```

- Ergebnis:

Cycles:	1.57137e+08 (-2,428%)
Instructions:	2.43771e+08 (-0,071%)
Branch Mispredictions:	1.49159e+06 (-13,044%)

Memcpy

- Transitions funktion
- Copy inst nach trans mit memcpy statt zuweisung

```
memcpy(trans, inst, sizeof(struct waypoint) * MAX_STATE);
```

- VS

```
for (k=MAX_STATE-1; k>=0; k--) {
    trans[k] = inst[k];
```

Memcpy (2)

- Ergebnis:

Cycles:	1.52184e+08 (-3,15%)
Instructions:	2.42871e+08 (-0,36%)
Mispredicted Branches:	1.2482 e+06 (-16,31%)

Ad Memcpy → Loop unrolled

- For-Schleife auf 8 Zeilen ausgerollt
- Ergebnis:

  Cycles: 1.63355e+08 (+7,34)

  Instructions: 2.40437e+08 (-1%)

  Mispredicted Branches: 1.75165e+06 (+40,33)
- Weniger Instructions, aber mehr Cycles →

  da mehr Branch Mispredictions
- Änderung verworfen

Inlining

- Cost functions mit inline keyword

  → `static` **`inline`** `int cost_nexts`

- Inlining trotz function pointer in struct?
- Ergebnis:
- Leicht verschlechter
- Gleiche Instructions
- Mehr Branch Mispredicitons

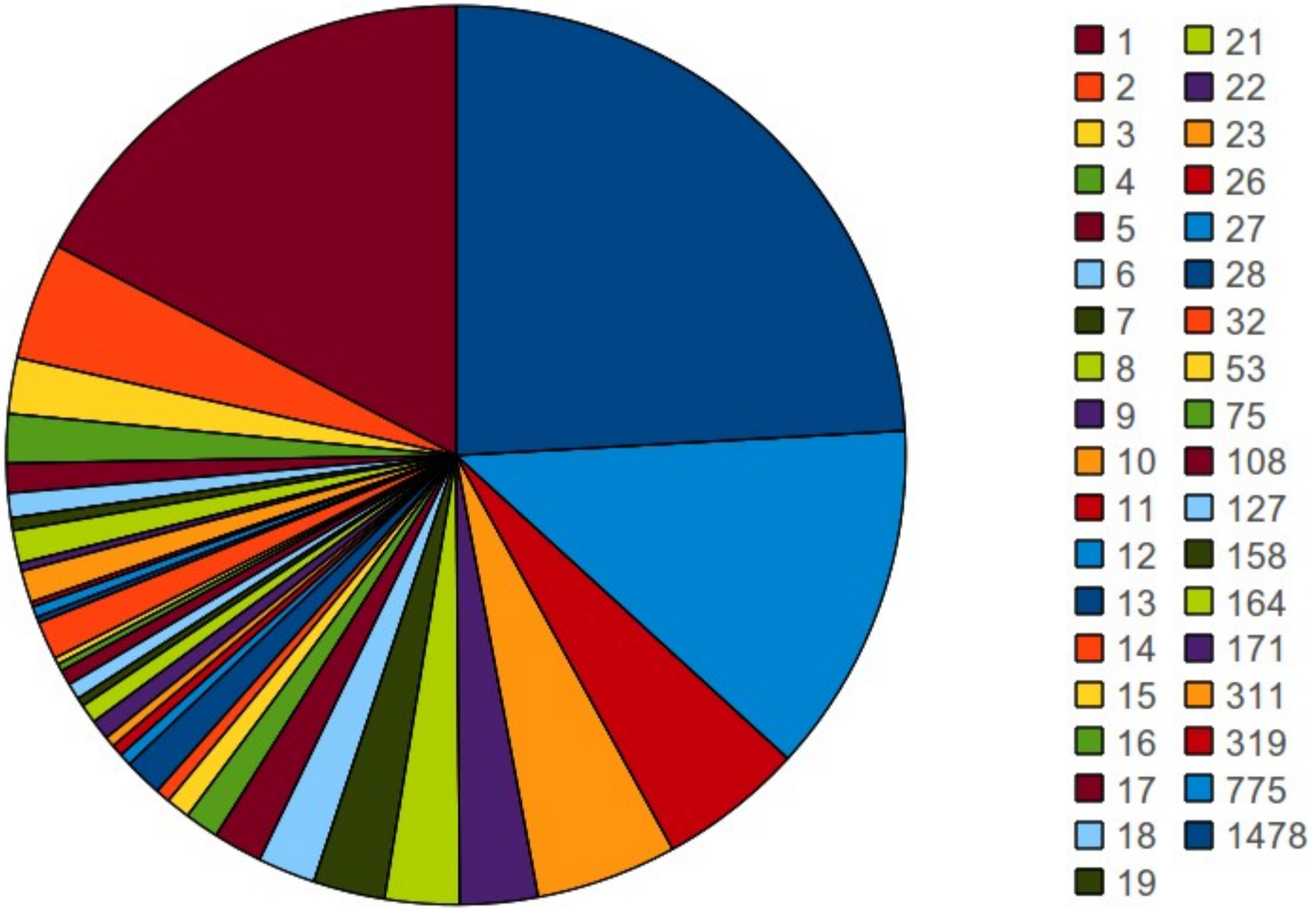
Memoization

- Input-Sequenz wiederholt sich
- Nicht ausreichend Ergebnis für jede PrimNum zu speichern
- Ein Sequenzblock hat immer selben Output

Memoization - Test-Daten

ffffffff a1000000 a0000000 0e000000	1_noop_0 0_lit@_1 1_@_;s_1
ffffffff a1000000 a0000000 0e000000	1_noop_0 0_lit@_1 1_@_;s_1
ffffffff a1000000 a0000000 0e000000	1_noop_0 0_lit@_1 1_@_;s_1
ffffffff a1000000 a0000000 0e000000	1_noop_0 0_lit@_1 1_@_;s_1
ffffffff 93000000 38000000 0b000000	1_noop_2 2_rot_3 3_+_2 2_noop_1 1_call_1
Ffffffff 93000000 38000000 0b000000	1_noop_2 2_rot_3 3_+_2 2_noop_1 1_call_1
ffffffff a1000000 a0000000 0e000000	1_noop_0 0_lit@_1 1_@_;s_1
ffffffff 0b000000	1_call_1
ffffffff a1000000	1_noop_0 0_lit@_1
ffffffff 0b000000	1_call_1
Ffffffff a1000000	1_noop_0 0_lit@_1
ffffffff a1000000 a0000000	1_noop_0 0_lit@_1 1_@_1
ffffffff 0e000000	1_;s_1
ffffffff	

Memoization - Analyse



Memoization - Trie

- 2 Durchläufe:
  - 1. baut Trie auf und zählt die Anzahl der Vorkomnisse der Sequenzblöcke
  - 2. berechnet Output und speichert diesen (wenn count > 1)
- Änderung von printinst:
  - Speichert entweder Wert in einem String oder gibt diesen aus.

Memoization - Ergebnis

Cycles:	7.88533e+07 (-48,185)
Instructions:	9.6525 e+07 (-60,256)
Mispredicted Branches:	880994 (-29,418)

Multi-Threading

- optimize_rewrite als Thread
- Quick'n'Dirty implementiert mit Pthreads
- Schlecht parallelisierbar → geteilte Datenstrukturen für waypoints

Multi-Threading (2)

- Minimaler Zeitgewinn?
- Realität: Thread overhead & schlechte implementierung
- Verbesserung:
  - waypoint arrays zeilenweise partitionieren
  - Locks auf Zeilen

Multi-Threading (3)

- Ergebnis:
- Time ohne Threads
  - real 0m0.058s
  - user 0m0.056s
  - sys 0m0.000s
- Time mit Threads
  - real 0m0.153s
  - user 0m0.024s
  - sys 0m0.044s

Compiler Optimierung
(In Klammern Rel. zu Original)

- Gcc -O Ergebnis:

  Cycles: 7.88533e+07 (-51%)

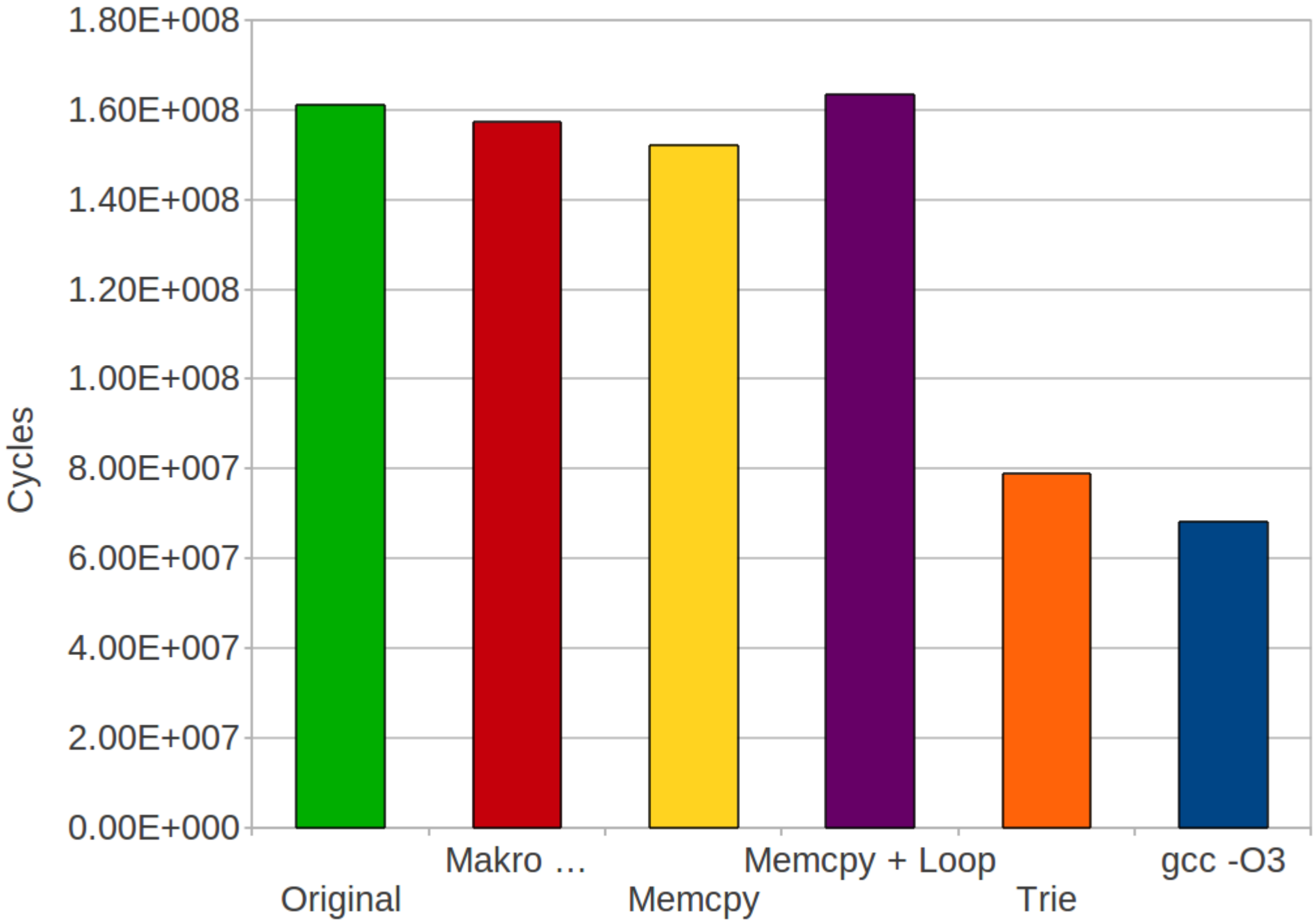
  Instructions: 9.6525 e+07 (-60,43%)

  Mispredicted Branches: 880994 (-48,64%)
- Gcc -O3 Ergebnis:

  Cycles: 6.8192 e+07 (-57,65%)

  Instructions: 9.11543e+07 (-62,63%)

  Mispredicted Branches: 785893 (-54,18%)


1.80E+008
1.60E+008
1.40E+008
1.20E+008
1.00E+008
8.00E+007
6.00E+007
4.00E+007
2.00E+007
0.00E+000
Cycles
Original
Makro …
Memcpy
Memcpy + Loop
Trie
gcc -O3