Protokoll zum Praktikum Parallelrechner - Übung 1

Fakultät Informatik
TU Dresden

Christian Kroh
Matrikelnummer: 3755154
Studiengang: Informatik (Diplom)
Jahrgang: 2010/2011

27. Januar 2014, Dresden

Inhaltsverzeichnis

1 Aufgabe 1 - Code-Optimierung

Listing 1: matmul0.c-Original

```
47        /* Begin matrix matrix multiply kernel */
48        for ( uint32_t i = 0; i < dim; i++ )
49        {
50            for ( uint32_t j = 0; j < dim; j++ )
51            {
52                for ( uint32_t k = 0; k < dim; k++ )
53                {
54                    // C[i][j] += A[i][k] * B[k][j]
55                    C[ i * dim + j ] += A[ i * dim + k ] * B[ k * dim + j ];
56                }
57            }
58        }
59        /* End matrix matrix multiply kernel */
```

1.1 Optimierungen

1. Speichern öfters genutzter, zusammengesetzter Werte
   `i_mult_dim = i * dim;`
   `i_mult_dim_add_k = i_mult_dim + k;`
   `k_mult_dim = k * dim;`

2. halten des Wertes einer Matrix für Multiplikation (loop-interchange)
   `A[ i_mult_dim_add_k ]`

Listing 2: matmul0.c-optimiert

```
46    uint32_t i_mult_dim, i_mult_dim_add_k, k_mult_dim, i, j, k;
47
48        /* Begin matrix matrix multiply kernel */
49        for ( i = 0; i < dim; i++ )
50        {
51      i_mult_dim = i * dim;
52            for ( k = 0; k < dim; k++ )
53            {
54        i_mult_dim_add_k = i_mult_dim + k;
55        k_mult_dim = k * dim;
56            for ( j = 0; j < dim; j++ )
57            {
58                    // C[i][j] += A[i][k] * B[k][j]
59                    C[ i_mult_dim + j ] += A[ i_mult_dim_add_k ] * B[ k_mult_dim + j ];
60                }
61            }
62        }
63        /* End matrix matrix multiply kernel */
```

2 Aufgabe 2 - Zeitmessungen

(nicht Taurus)

2.1 Original, gcc, keine flags

Dimension	Runtime	GFLOP/s
32	0.0002s	0.40
64	0.0015s	0.40
96	0.0045s	0.40
128	0.0115s	0.38
160	0.0217s	0.37
192	0.0503s	0.36
224	0.0651s	0.35
256	0.1365s	0.25
320	0.1984s	0.33
384	0.4676s	0.25
448	0.5567s	0.33
512	1.3026s	0.21
640	2.9606s	0.18
768	5.5037s	0.16
896	8.5051s	0.17
1024	26.0022s	0.16

2.2 Original, gcc, -O3

Dimension	Runtime	GFLOP/s
32	0.0000s	1.73
64	0.0003s	1.85
96	0.0009s	1.93
128	0.0031s	1.35
160	0.0055s	1.56
192	0.0104s	1.37
224	0.0159s	1.42
256	0.0339s	1.00
320	0.0537s	1.24
384	0.1077s	1.06
448	0.1524s	1.18
512	0.3500s	0.78
640	1.6626s	0.33
768	3.2088s	0.29
896	5.1806s	0.28
1024	7.5068s	0.29

Tabelle 1: Original, gcc

2.3 Mit Optimierungen, gcc, keine flags

Dimension	Runtime	GFLOP/s
32	0.0001s	0.61
64	0.0009s	0.61
96	0.0026s	0.70
128	0.0058s	0.71
160	0.0115s	0.71
192	0.0198s	0.71
224	0.0313s	0.72
256	0.0465s	0.72
320	0.0902s	0.73
384	0.1552s	0.73
448	0.2451s	0.73
512	0.3632s	0.74
640	0.7120s	0.74
768	1.2261s	0.74
896	1.9538s	0.74
1024	2.9417s	0.73

2.4 Mit Optimierungen, gcc, -O3

Dimension	Runtime	GFLOP/s
32	0.0000s	2.12
64	0.0002s	2.24
96	0.0006s	2.50
128	0.0015s	2.58
160	0.0031s	2.56
192	0.0052s	2.65
224	0.0081s	2.67
256	0.0120s	2.75
320	0.0239s	2.72
384	0.0408s	2.76
448	0.0647s	2.77
512	0.0941s	2.86
640	0.1886s	2.78
768	0.3226s	2.81
896	0.5146s	2.80
1024	0.7530s	2.85

Tabelle 2: Mit Optimierungen, gcc

2.5 Original, icc, keine flags

Dimension	Runtime	GFLOP/s
32	0.0000s	1.52
64	0.0001s	3.67
96	0.0003s	4.30
128	0.0008s	4.88
160	0.0018s	4.87
192	0.0030s	4.90
224	0.0046s	4.73
256	0.0072s	4.73
320	0.0125s	4.99
384	0.0228s	5.02
448	0.0346s	5.17
512	0.0533s	5.02
640	0.1020s	5.13
768	0.1739s	5.15
896	0.2743s	5.25
1024	0.4137s	5.19

2.6 Original, icc, -O3

Dimension	Runtime	GFLOP/s
32	0.0000s	1.98
64	0.0001s	3.69
96	0.0004s	4.00
128	0.0009s	4.51
160	0.0016s	4.67
192	0.0031s	4.60
224	0.0051s	4.43
256	0.0076s	4.40
320	0.0144s	4.57
384	0.0245s	4.66
448	0.0386s	4.67
512	0.0582s	4.60
640	0.1117s	4.70
768	0.1937s	4.69
896	0.3035s	4.75
1024	0.4552s	4.73

Tabelle 3: Original, icc

2.7 Mit Optimierungen, icc, keine flags

Dimension	Runtime	GFLOP/s
32	0.0000s	2.99
64	0.0001s	5.09
96	0.0003s	5.41
128	0.0007s	5.98
160	0.0014s	5.95
192	0.0024s	5.65
224	0.0041s	5.41
256	0.0062s	5.29
320	0.0121s	5.45
384	0.0202s	5.52
448	0.0327s	5.57
512	0.0481s	5.61
640	0.0928s	5.63
768	0.1606s	5.65
896	0.2522s	5.70
1024	0.3757s	5.72

2.8 Mit Optimierungen, icc, -O3

Dimension	Runtime	GFLOP/s
32	0.0000s	2.72
64	0.0001s	5.07
96	0.0003s	5.34
128	0.0006s	5.67
160	0.0016s	5.60
192	0.0024s	5.78
224	0.0046s	5.35
256	0.0062s	5.32
320	0.0122s	5.44
384	0.0202s	5.51
448	0.0322s	5.58
512	0.0482s	5.57
640	0.0950s	5.52
768	0.1650s	5.50
896	0.2561s	5.54
1024	0.3841s	5.58

Tabelle 4: Mit Optimierungen, icc

3 Aufgabe 3 - Compiler-Flags

3.1 -O3

Bei Verwendung des gcc-Compilers bringt dieser Flag eine Verbesserung der Ausführungszeit vom Faktor 4 mit sich. Allerdings verschlechtert er die Ausführungszeit beim icc.

3.2 -floop-interchange

Führt eine Vertauschung von Schleifen aus, ähnlich wie die Code-Optimierung.

3.3 -funroll-loops

Nimmt Schleifen auseinander, deren Schritte durch den Compiler vor der Ausführung bestimmt werden können.

4 Aufgabe 4 - FLOP/s

Intel E5-2690 Sandy Bridge

Maximaler (Daten-)Cache per Core: 20MB L3 + 256KB L2 + 32KB L1 = 20,35 MB

FLOP/s je Kern (Normal Clock): $2,9GHz * \frac{8\ FLOPs}{cycle} = 23,2\ GFLOP/s\ per\ Core$

Notwendige Speicherbandbreite (für einen Kern): $23,2\ GFLOP/s * 32\frac{Bit}{FLOP} = 92,8\ GB/s$

FLOP/s je Kern (Maximum Clock): $3,8GHz * \frac{8\ FLOPs}{cycle} = 30,4\ GFLOP/s\ per\ Core$

Notwendige Speicherbandbreite (für einen Kern): $30,4\ GFLOP/s * 32\frac{Bit}{FLOP} = 121,6\ GB/s$

L1-Cache: 32KB Daten, 4 Takte Latenz, Tatsächliche Speicherbandbreite (ein Kanal, DDR3-1600): $1600MT/s * 8Bit = 12,8\ GB/s$

Verbrauchter Speicher (3 Matritzen mit jeweils 1 Mio Einträgen): $3 * 1000000 * 32 = 12MB$

Dass maximal in etwa ein Viertel (ca. 6 GFLOP/s) der möglichen 23,2 GFLOP/s erreicht wird, liegt an der Cache-Latenz. Da der benötigte Speicherplatz die Kapazität der Caches nicht überschreitet, müssen keine Daten aus dem RAM geladen werden.
Jedoch können pro Takt nur 4 Floats aus dem L1,2 Floats aus L2 und 1 Float aus L3 geladen werden, was maximal 7 Floats per Takt liefert (bei einer idealen Verteilung). Um tatsächlich alle FPUs des Kerns auszulasten, müssten pro Operand mehrere Operationen ausgeführt werden.