



- Wie parallelisiere ich?
- Was kann ich parallelisieren?
- Wann macht es Sinn?

1. die letzten paar Themen gingen auch um Parallelisierung
2. Parallelisierung voll toll, wie geht man das an

*Zukunft in  
Bewegung*

# *EMB<sup>2</sup>*

## *Vergleich zu OpenMP*

Simon Varga

8. Juni 2015

2015-06-08 EMB<sup>2</sup>

1. Embedded Multicore Building Blocks
2. kein Paper, sondern API





- 1 Parallelisierung
- 2 OpenMP
- 3 EMB<sup>2</sup>
- 4 Beispiele
- 5 Ergebnis

2015-06-08 EMB<sup>2</sup>

└ Inhalt

1. ganz kurz nochmal zur Parallelisierung
2. kurze Vorstellung von OpenMP
3. danach auch EMB2 und Unterschiede zu OpenMP
4. Beispiele in OpenMP und EMB2
5. Am Schluss Ergebnis

- Parallelisierung
- OpenMP
- EMB<sup>2</sup>
- Beispiele
- Ergebnis

# Parallelisierung

## Wieso macht man das?



- Multicore-Systeme
- Schnelle Ausführung

2015-06-08

EMB<sup>2</sup>

- └ Parallelisierung
  - └ Wieso macht man das?
    - └ Parallelisierung

Parallelisierung  
Wieso macht man das?

- Multicore-Systeme
- Schnelle Ausführung

1. auch in Autos, kleine Gebrauchsgegenstände
2. Prozessorgeschwindigkeit steigt nicht mehr, Anzahl dagegen schon

# Parallelisierung

## Was kann ich parallelisieren?



2015-06-08

EMB<sup>2</sup>

- └ Parallelisierung
  - └ Was kann ich parallelisieren?
    - └ Parallelisierung

Parallelisierung  
Was kann ich parallelisieren?

- Sequentieller ↔ Paralleler Anteil
- Logische Unabhängigkeit

- Sequentieller ↔ Paralleler Anteil
- Logische Unabhängigkeit

1. identifizierung
2. verschieden unabhängige Bereiche im Programm



src/hello.cpp

```
1 int main() {  
2     std::cout << "Hello World!" << std::endl;  
3 }
```

2015-06-08

EMB<sup>2</sup>

- └ Parallelisierung
- └ Beispiele
- └ Parallelisierung

src/hello.cpp

```
1 int main() {  
2     std::cout << "Hello World!" << std::endl;  
3 }
```

1. kurzes Programm, stellvertretend für viele Sequenzielle Codezeilen
- 2.
3. nicht oder nur sehr schlecht zu Parallelisieren



src/hello.cpp

```
1 int main() {  
2     std::cout << "Hello World!" << std::endl;  
3 }
```

2015-06-08

EMB<sup>2</sup>

- └ Parallelisierung
- └ Beispiele
- └ Parallelisierung



1. kurzes Programm, stellvertretend für viele Sequenzielle Codezeilen
- 2.
3. nicht oder nur sehr schlecht zu Parallelisieren



## src/section.cpp

```
1 int main() {  
2     std::ifstream file1("file1.txt");  
3     std::string line1;  
4     std::getline(file1, line1);  
5     std::cout << "File1: " << line1 << std::endl;  
6  
7     std::ifstream file2("file2.txt");  
8     std::string line2;  
9     std::getline(file2, line2);  
10    std::cout << "File2: " << line2 << std::endl;  
11  
12    return 0;  
13 }
```

2015-06-08

EMB<sup>2</sup>

└ Parallelisierung

└ Beispiele

└ Parallelisierung

```
src/section.cpp  
1 int main() {  
2     std::ifstream file1("file1.txt");  
3     std::string line1;  
4     std::getline(file1, line1);  
5     std::cout << "File1: " << line1 << std::endl;  
6  
7     std::ifstream file2("file2.txt");  
8     std::string line2;  
9     std::getline(file2, line2);  
10    std::cout << "File2: " << line2 << std::endl;  
11  
12    return 0;  
13 }
```

1. 1. Zeile von Datei ausgeben
- 2.
3. unabhängige Bereiche, trennen





## src/section.cpp

```
1 int main() {  
2     std::ifstream file1("file1.txt");  
3     std::string line1;  
4     std::getline(file1, line1);  
5     std::cout << "File1: " << line1 << std::endl;  
6  
7     std::ifstream file2("file2.txt");  
8     std::string line2;  
9     std::getline(file2, line2);  
10    std::cout << "File2: " << line2 << std::endl;  
11  
12    return 0;  
13 }
```

```
src/section.cpp  
1 int main() {  
2     std::ifstream file1("file1.txt");  
3     std::string line1;  
4     std::getline(file1, line1);  
5     std::cout << "File1: " << line1 << std::endl;  
6  
7     std::ifstream file2("file2.txt");  
8     std::string line2;  
9     std::getline(file2, line2);  
10    std::cout << "File2: " << line2 << std::endl;  
11  
12    return 0;  
13 }
```

1. 1. Zeile von Datei ausgeben
- 2.
3. unabhängige Bereiche, trennen



## src/loop.cpp

```
1 int main() {  
2     for (unsigned int i = 0; i < 10; ++i) {  
3         std::cout << "i = " << i << std::endl;  
4     }  
5 }
```

2015-06-08

EMB<sup>2</sup>

└ Parallelisierung

└ Beispiele

└ Parallelisierung

```
src/loop.cpp  
1 int main() {  
2     for (unsigned int i = 0; i < 10; ++i) {  
3         std::cout << "i = " << i << std::endl;  
4     }  
5 }
```

1. Schleife (in der was ausgegeben wird)
- 2.
3. Aufteilung auf beliebig viele unabhängige Einzelteile

src/loop.cpp

```
1 int main() {  
2     for (unsigned int i = 0; i < 10; ++i) {  
3         std::cout << "i = " << i << std::endl;  
4     }  
5 }
```

2015-06-08

EMB<sup>2</sup>

└ Parallelisierung

└ Beispiele

└ Parallelisierung

src/loop.cpp

```
1 int main() {  
2     for (unsigned int i = 0; i < 10; ++i) {  
3         std::cout << "i = " << i << std::endl;  
4     }  
5 }
```

1. Schleife (in der was ausgegeben wird)
- 2.
3. Aufteilung auf beliebig viele unabhängige Einzelteile



## src/prefix\_computation.cpp

```
1 int main() {  
2     std::vector<unsigned int> range(11);  
3     range[0] = 0;  
4  
5     for (unsigned int i = 1; i < range.size(); ++i) {  
6         range[i] = range[i - 1] + 1;  
7     }  
8 }
```

2015-06-08

EMB<sup>2</sup>

└ Parallelisierung

└ Beispiele

└ Parallelisierung

src/prefix\_computation.cpp

```
1 int main() {  
2     std::vector<unsigned int> range(11);  
3     range[0] = 0;  
4  
5     for (unsigned int i = 1; i < range.size(); ++i) {  
6         range[i] = range[i - 1] + 1;  
7     }  
8 }
```

1. Schleife, aber abhängig von vorherigem Ergebnis
2. Schwierig, nicht von allen Unterstützt

# Parallelisierung

## Was nun?



2015-06-08

EMB<sup>2</sup>

- └ Parallelisierung
  - └ Was nun?
    - └ Parallelisierung

Parallelisierung  
Was nun?

- Programmstruktur identifizieren
- Geeignete Mittel einsetzen
- Anpassung an Zielsystem

- Programmstruktur identifizieren
- Geeignete Mittel einsetzen
- Anpassung an Zielsystem

1. Wie ist das Programm aufgeteilt
2. Hängt ab vom Ziel, Multiplattform, Embedded UND vom Programm selber, ist es schon fertig, wird ein neues geschrieben
3. Auf Eigenheiten des Systems eingehen, evtl. sogar auf 2,3,4 Prozessoren fest einstellen

# Parallelisierung

## Was setze ich ein?



- fork
- Compilerflag (-ftree-parallelize-loops)
- PThread, std::thread, ...
- OpenMP
- EMB<sup>2</sup>

2015-06-08

EMB<sup>2</sup>

- └ Parallelisierung
  - └ Was setze ich ein?
    - └ Parallelisierung

1. sollte jeder kennen, (komplett) getrennte Prozesse
2. für C/C++ nur Schleifen, auch nicht alle, aber man muss nur flag + pthread linken und kann es ausprobieren
3. Standardwerkzeuge, C++-11
4. gleich mehr
5. was kann das, kleiner Vergleich

Parallelisierung  
Was setze ich ein?

- fork
- Compilerflag (-ftree-parallelize-loops)
- PThread, std::thread, ...
- OpenMP
- EMB<sup>2</sup>

# OpenMP

## Wie funktioniert das?



- Für C/C++, Fortran
- ähnlich mächtig zu PThreads, std::thread
- Präprozessormakros (#pragma)
- integriert in Compiler

2015-06-08

EMB<sup>2</sup>

└ OpenMP

└ Wie funktioniert das?

└ OpenMP

1. Aktuell bei gcc-5.\* OpenMP 4.0
- 2.
3. Steuerung, Angabe wie parallelisiert werden soll, Definition von Compileroptionen, wenn nicht ausgewertet werden kann, einfach ignoriert
4. Compiler muss unterstützen, aber alle gängigen

OpenMP  
Wie funktioniert das?

- Für C/C++, Fortran
- ähnlich mächtig zu PThreads, std::thread
- Präprozessormakros (#pragma)
- integriert in Compiler

# OpenMP

## Wie sieht das aus?



src/openmp.cpp

```
1 #include <iostream>
2 #include <omp.h>
3
4 int main() {
5     int max = 10;
6
7     #pragma omp parallel for //num_threads(2)
8     for (int i = 0; i < max; i++) {
9         std::cout << "Done in Thread "
10                 << omp_get_thread_num()
11                 << ", num = " << i << std::endl;
12     }
13
14     return 1;
15 }
```

2015-06-08

EMB<sup>2</sup>

└ OpenMP

└ Wie sieht das aus?

└ OpenMP

OpenMP  
Wie sieht das aus?

```
src/openmp.cpp
1 #include <iostream>
2 #include <omp.h>
3
4 int main() {
5     int max = 10;
6
7     #pragma omp parallel for //num_threads(2)
8     for (int i = 0; i < max; i++) {
9         std::cout << "Done in Thread "
10                 << omp_get_thread_num()
11                 << ", num = " << i << std::endl;
12     }
13
14     return 1;
15 }
```

kleine Beispielanwendung

1. {2} OpenMP Header
2. {7} Magic, man kann die Anzahl der Threads angeben
3. {10} auf Threadnummer zugreifen





## src/openmp.out

```
1 Done in Thread Done in Thread Done in Thread 10, num = 33,  
   num = , num = 80Done in Thread  
2  
3  
4 2, num = 6Done in Thread 1, num = 4  
5 Done in Thread 1, num = 5  
6  
7 Done in Thread 2, num = 7  
8 Done in Thread 3, num = 9  
9 Done in Thread 0, num = 1  
10 Done in Thread 0, num = 2
```

2015-06-08

EMB<sup>2</sup>

└ OpenMP

└ Wie sieht das aus?

└ OpenMP

```
src/openmp.out  
1 Done in Thread Done in Thread Done in Thread 10, num = 33,  
   num = , num = 80Done in Thread  
2  
3  
4 2, num = 6Done in Thread 1, num = 4  
5 Done in Thread 1, num = 5  
6  
7 Done in Thread 2, num = 7  
8 Done in Thread 3, num = 9  
9 Done in Thread 0, num = 1  
10 Done in Thread 0, num = 2
```

Output bei 4 Threads, mehrere Sachen sehen:

1. mehrere schreiben gleichzeitig, durcheinander, leerzeilen
2. nicht in fester Reihenfolge
- 3.
4. Wie siehts mit EMB2 aus



- von Siemens entwickelt
- vor allem für Embedded Systeme
- C/C++
- Ziel: Abstraktion von (low-Level) Thread-Management
- Unterstützt Task Prioritäten
- Aufgebaut auf MTAPI
  - standardisiertes Programminterface
  - Unterstützt symmetrische und asymmetrische Prozessoren

2015-06-08

EMB<sup>2</sup>

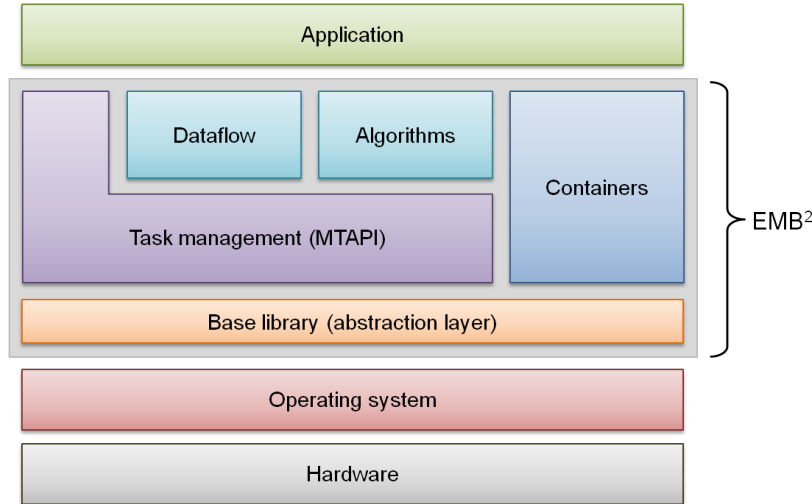
└ EMB<sup>2</sup>

└ Embedded Multicore Building Blocks

└ EMB<sup>2</sup>

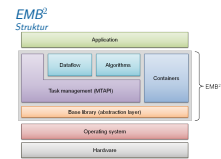
- von Siemens entwickelt
- vor allem für Embedded Systeme
- C/C++
- Ziel: Abstraktion von (low-Level) Thread-Management
- Unterstützt Task Prioritäten
- Aufgebaut auf MTAPI
  - standardisiertes Programminterface
  - Unterstützt symmetrische und asymmetrische Prozessoren

1. ab 1.10.2014 Open Source, aktuell 0.3.0 vom 27. Mai, auf github
2. aber auch für normale Software geeignet
- 3.
4. man muss sich nicht mehr selber drum kümmern
5. hat geheißen, das machen nicht viele
6. Multicore Task Management API
7. um Parallelisierung auf Embedded Systeme zu bringen
8. asymmetrische: nicht alle haben gleiche Priorität/Aufgabenverteilung, z.B. nur einer kann Betriebssystem-Code ausführen



2015-06-08

EMB<sup>2</sup>  
└ EMB<sup>2</sup>  
└ Struktur  
└ EMB<sup>2</sup>



1. oben Applikation, die die Bibliothek benutzt
2. eigentliche Bib, setzt auf MTAPI für die Taskverwaltung auf
3. bietet zusätzliche Klassen und Funktionen, für viel genutzte Algorithmen
4. bietet auch eigene Containerklassen, deren Funktionen parallel bearbeitet werden
5. läuft auf den verschiedensten Prozessorarchitekturen (x86, ARM)



2015-06-08

EMB<sup>2</sup>  
└ EMB<sup>2</sup>  
  └ Beispiel  
    └ EMB<sup>2</sup>

Code

kommt gleich danach → Codeblocks



- + neue Software
- + Algorithmen, Datenstrukturen
  - vorhandene Software

2015-06-08

EMB<sup>2</sup>

└ EMB<sup>2</sup>

└ Unterschied zu OpenMP

└ EMB<sup>2</sup>

1. wenn Software neu geschrieben wird
2. bietet Standardalgorithmen und Datenstrukturen
3. relativ viel umschreiben



Code

2015-06-08

EMB<sup>2</sup>

└ Beispiele

└ Beispiele

Beispiele

Code

in Codeblocks



2015-06-08

EMB<sup>2</sup>

└ Ergebnis  
└ Aufwand  
└ Ergebnis

Ergebnis  
Aufwand

- Compilerflag sinnlos
- OpenMP relativ einfach
- EMBB für neue Programme

- Compilerflag sinnlos
- OpenMP relativ einfach
- EMBB für neue Programme

1. bei meinen Beispielen hat es nichts gebracht
2. bei bestehenden Programmen
3. etwas schwierig die genau gleichen Beispiele, wenn man eh neu schreibt, kann man die beste Logik benutzen



2015-06-08

EMB<sup>2</sup>

└ Ergebnis  
└ Zeiten  
└ Ergebnis

Ergebnis  
Zeiten

	Std	OpenMP	SpeedUp	EMBB	SpeedUp
Schleife o. Sleep	24 $\mu s$	4.423 $\mu s$	0,005	95 $\mu s$	0,25
Schleife	1.000.957 $\mu s$	315.302 $\mu s$	3,17	401.424 $\mu s$	2,45
Abh. Schleife	900.662 $\mu s$	303.323 $\mu s$	2,97	402.850 $\mu s$	2,24
Reduce	1.000.736 $\mu s$	301.900 $\mu s$	3,13	703.606 $\mu s$	1,42
Reduce Custom	1.000.742 $\mu s$	301.928 $\mu s$	3,15	704.277 $\mu s$	1,42
Section	200.231 $\mu s$	100.102 $\mu s$	2,00	100.185 $\mu s$	2,00
Sort	1.901.300 $\mu s$	1.609.059 $\mu s$	1,18	3.302.131 $\mu s$	0,58

	Std	OpenMP	SpeedUp	EMBB	SpeedUp
Schleife o. Sleep	24 $\mu s$	4.423 $\mu s$	0,005	95 $\mu s$	0,25
Schleife	1.000.957 $\mu s$	315.302 $\mu s$	3,17	401.424 $\mu s$	2,45
Abh. Schleife	900.662 $\mu s$	303.323 $\mu s$	2,97	402.850 $\mu s$	2,24
Reduce	1.000.736 $\mu s$	301.900 $\mu s$	3,13	703.606 $\mu s$	1,42
Reduce Custom	1.000.742 $\mu s$	301.928 $\mu s$	3,15	704.277 $\mu s$	1,42
Section	200.231 $\mu s$	100.102 $\mu s$	2,00	100.185 $\mu s$	2,00
Sort	1.901.300 $\mu s$	1.609.059 $\mu s$	1,18	3.302.131 $\mu s$	0,58

1. Zeiten verglichen, natürlich nur künstlich
2. jeweils mit 4 Threads
3. möglicherweise bei EMBB sleep nicht sauber → wird möglicherweise optimiert





2015-06-08

EMB<sup>2</sup>

└ Ergebnis

└ Zeiten

Vielen Dank für die Aufmerksamkeit  
Ich habe fertig

Vielen Dank für die Aufmerksamkeit  
Ich habe fertig