Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

DawnCC Automatisierte Parallelisierung

Seminar - Effiziente Programmierung - WiSe19/20

Anton Kollewe (6988354)

26.11.2019

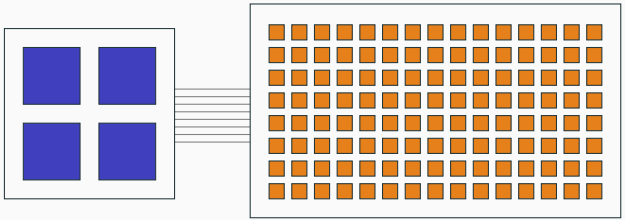
Universität Hamburg

Agenda

- Was ist DawnCC?
- Was ist OpenACC / OpenMP?
- Beispiel
- Wie funktioniert DawnCC?
  - Memory bound estimation
  - Parallelisierbare Schleifen ermitteln
  - Pointer restrictification
- Vor- & Nachteile
- Zusammenfassung

Was ist DawnCC?

- Source-to-Source compiler
- Automatisiertes Einfügen von Parallelisierungs- und Offloading-Direktiven
  → OpenACC / OpenMP



- Static analysis
- Sammlung von LLVM passes

Was ist OpenACC / OpenMP?

- Parallelisierung und Offloading sequentiellen Codes
- Erweiterung der C / C++ / Fortran Standards
  - → Compiler-Direktiven (Meta-Sprache)
  - → Funktionsbibliothek
  - → Umgebungsvariablen
- Inkrementeller Übergang
- Portable, auch single-core

- OpenMP: Explizite Parallelisierung
- OpenACC: Implizite Optimierung

OpenMP Beispiel

```
1 void simple(int a) {
2     int n[100];
3
4
5     for (int i = 0; i < a; ++i) {
6         n[i] += 1;
7     }
8 }
```

OpenMP Beispiel

```
1 void simple(int a) {
2     int n[100];
3     #pragma omp target data map(tofrom: n[0:100])
4     #pragma omp parallel for
5     for (int i = 0; i < a; ++i) {
6         n[i] += 1;
7     }
8 }
```

OpenACC Beispiel

```
1 void simple(int a) {
2     int n[100];
3     #pragma acc data pcopyin(n[0:100])
4     #pragma acc kernels loop independent
5     for (int i = 0; i < a; ++i) {
6         n[i] += 1;
7     }
8 }
```

OpenACC Beispiel

```
1 void simple(int a) {
2     int n[100];
3     #pragma acc data pcopyin(n[0:100]) if(a > 10)
4     #pragma acc kernels loop independent if(a > 10)
5     for (int i = 0; i < a; ++i) {
6         n[i] += 1;
7     }
8 }
```

Application developers are responsible for correctly using the OpenMP API to produce a conforming program.

The OpenMP API does not cover compiler-generated automatic parallelization.

- OpenMP 5.0 Specification

DawnCC Beispiel: saxpy

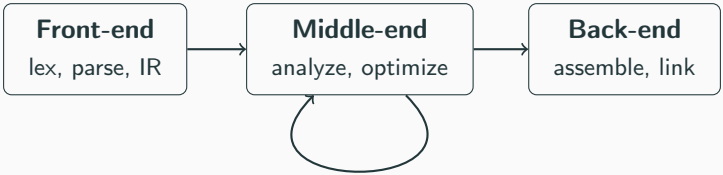
```
1 void saxpy(float a, float* x, float* y, int n) {
2   for (int i = 0; i < n; ++i) {
3     y[i] = a * x[i] + y[i];
4   }
5 }
```

DawnCC Beispiel: saxpy

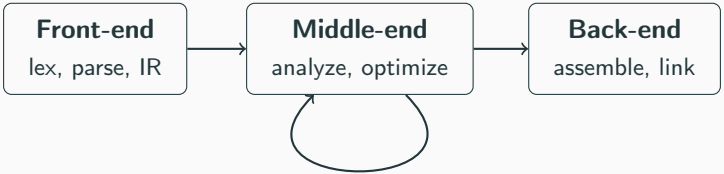
```
1  void saxpy(float a, float* x, float* y, int n) {
2    long long int tmp;
3    tmp = n - 1;
4    char x_y_alias_free = ((x > y + tmp) ||
5                           (y > x + tmp));
6    #pragma omp target data \
7                map(to:x[0:tmp+1]) map(tofrom:y[0:tmp+1]) \
8                if(x_y_alias_free)
9    #pragma omp parallel for if(x_y_alias_free)
10   for (int i = 0; i < n; ++i) {
11     y[i] = a * x[i] + y[i];
12   }
13 }
```

Wie funktioniert DawnCC?

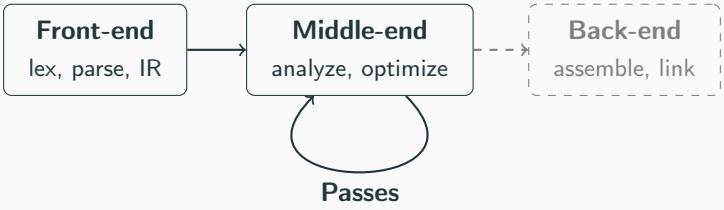
LLVM Passes


Front-end
lex, parse, IR
Middle-end
analyze, optimize
Back-end
assemble, link

LLVM Passes


Front-end
lex, parse, IR
Middle-end
analyze, optimize
Back-end
assemble, link
Passes
Module Pass,
Function Pass,
Loop Pass,
Region Pass,
...

LLVM Passes


Front-end
lex, parse, IR
Middle-end
analyze, optimize
Back-end
assemble, link
Passes
Module Pass,
Function Pass,
Loop Pass,
Region Pass,
...

Memory Bound Estimation

- Kopieren von Speicher auf Beschleuniger (und zurück)
- Benötigt Größenschätzung des benutzten Speichers

→ Symbolic Range Analysis

  - Wertebereich eines Integers
  - Upper & Lower bound

- Zugriffe auf Array

Symbolic Range Analysis

- Kompiliere zu LLVM IR
- Finde Range von Symbol `i`: $[l(\texttt{i}), u(\texttt{i})]$

$\rightarrow$ Für jede LLVM IR instruction: Was passiert mit den bounds?

$\rightarrow$ Branches: Vereinigung

$\rightarrow$ Konvergenz durch Widening

Symbolic Range Analysis

- Kompiliere zu LLVM IR
- Finde Range von Symbol `i`: $[l(\texttt{i}), u(\texttt{i})]$

$\rightarrow$ Für jede LLVM IR instruction: Was passiert mit den bounds?

$\rightarrow$ Branches: Vereinigung

$\rightarrow$ Konvergenz durch Widening

- Symbolic Range $\rightarrow$ Memory bound: Zugriffssymbole
  - Array `x` mit basepointer `x[0]`
  - Range von `i`: $[l(\texttt{i}), u(\texttt{i})]$
  - Bounds von `x[i]`: $[\texttt{x[0]} + l(\texttt{i}), \texttt{x[0]} + u(\texttt{i})]$

Symbolic Range Analysis: Beispiel

```
1  void test(int x) {
2      int y;
3      y = 3;
4      ++y;
5      if (x >= -5) {
6          if (x <= 2) {
7              y = y + x;
8              A[y]...
9          }
10     }
11 }
```

Symbolic Range Analysis: Beispiel

```
→  1  void test(int x) {
   2      int y;
   3      y = 3;
   4      ++y;
   5      if (x >= -5) {
   6          if (x <= 2) {
   7              y = y + x;
   8              A[y]...
   9          }
  10      }
  11  }
```

x: $[-\infty, +\infty]$

Symbolic Range Analysis: Beispiel

```
1  void test(int x) {
2      int y;
3      y = 3;
4      ++y;
5      if (x >= -5) {
6          if (x <= 2) {
7              y = y + x;
8              A[y]...
9          }
10     }
11 }
```

x: $[-\infty, +\infty]$
y: $[-\infty, +\infty]$

Symbolic Range Analysis: Beispiel

```
1  void test(int x) {
2      int y;
3      y = 3;
4      ++y;
5      if (x >= -5) {
6          if (x <= 2) {
7              y = y + x;
8              A[y]...
9          }
10     }
11 }
```

x: $[-\infty, +\infty]$
y: $[3, 3]$

Symbolic Range Analysis: Beispiel

```
1  void test(int x) {
2      int y;
3      y = 3;
→ 4    ++y;
5      if (x >= -5) {
6          if (x <= 2) {
7              y = y + x;
8              A[y]...
9          }
10     }
11 }
```

x: $[-\infty, +\infty]$
y: $[4, 4]$

Symbolic Range Analysis: Beispiel

```
1  void test(int x) {
2      int y;
3      y = 3;
4      ++y;
→ 5    if (x >= -5) {
6          if (x <= 2) {
7              y = y + x;
8              A[y]...
9          }
10     }
11 }
```

x: $[-5, +\infty]$
y: $[4, 4]$

Symbolic Range Analysis: Beispiel

```
 1 void test(int x) {
 2     int y;
 3     y = 3;
 4     ++y;
 5     if (x >= -5) {
→ 6         if (x <= 2) {
 7             y = y + x;
 8             A[y]...
 9         }
10     }
11 }
```

x: $[-5, 2]$
y: $[4, 4]$

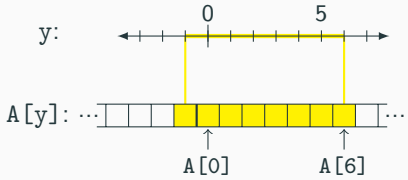
Symbolic Range Analysis: Beispiel

```
1  void test(int x) {
2      int y;
3      y = 3;
4      ++y;
5      if (x >= -5) {
6          if (x <= 2) {
→ 7              y = y + x;
8              A[y]...
9          }
10     }
11 }
```

x: $[-5, 2]$
y: $[-1, 6]$

Symbolic Range Analysis: Beispiel

```
1  void test(int x) {
2      int y;
3      y = 3;
4      ++y;
5      if (x >= -5) {
6          if (x <= 2) {
7              y = y + x;
→ 8            A[y]...
9          }
10     }
11 }
```



Parallelisierbare Schleifen

- Welche Schleifen können annotiert werden?
- Spezialfall:
  - Sämtliche Datenabhängigkeiten innerhalb des Rumpfes
  - Induktionsvariable steigt linear mit Steigung $\beta > 0$
  - Array-Zugriffe Range maximal β breit

- Mehr existieren

$\rightarrow$ Nicht von DawnCC abgedeckt

Parallelisierbare Schleifen

- Welche Schleifen können annotiert werden?
- Spezialfall:
  - Sämtliche Datenabhängigkeiten innerhalb des Rumpfes
  - Induktionsvariable steigt linear mit Steigung $\beta > 0$
  - Array-Zugriffe Range maximal β breit

```
1 for (int i = 1; i < n; i += 2) {
2     int tmp = log(i);
3     A[i] = A[i-1] + A[i] + tmp;
4 }
```

$\rightarrow$ Parallelisierbar

Parallelisierbare Schleifen

- Welche Schleifen können annotiert werden?
- Spezialfall:
  - Sämtliche Datenabhängigkeiten innerhalb des Rumpfes
  - Induktionsvariable steigt linear mit Steigung $\beta > 0$
  - Array-Zugriffe Range maximal β breit

```
1 for (int i = 1; i < n; i += 2) {
2     A[i] = A[i-1] + A[i+1];
3 }
```

$\rightarrow$ Nicht parallelisierbar

Parallelisierbare Schleifen

- Welche Schleifen können annotiert werden?
- Spezialfall:
  - Sämtliche Datenabhängigkeiten innerhalb des Rumpfes
  - Induktionsvariable steigt linear mit Steigung $\beta > 0$
  - Array-Zugriffe Range maximal β breit

```
1 for (int i = 0; i < n; ++i) {
2     y[i] = a * x[i] + y[i];
3 }
```

$\rightarrow$ Parallelisierbar

Parallelisierbare Schleifen

- Welche Schleifen können annotiert werden?
- Spezialfall:
  - Sämtliche Datenabhängigkeiten innerhalb des Rumpfes
  - Induktionsvariable steigt linear mit Steigung $\beta > 0$
  - Array-Zugriffe Range maximal β breit

```
1 for (int i = 0; i < n; ++i) {
2     y[i] = a * x[i] + y[i];
3 }
```

$\rightarrow$ Parallelisierbar, wenn ...

Pointer Restrictification

- Vermeiden von pointer aliasing

→ Wenn memory bounds nicht überschneiden

Kein aliasing von pointern p und q mit ranges $[l_p, u_p]$ und $[l_q, u_q]$ wenn:
$p + l_p > q + u_q$ oder $q + l_q > p + u_p$.

Pointer Restrictification

- Vermeiden von pointer aliasing

→ Wenn memory bounds nicht überschneiden

Kein aliasing von pointern p und q mit ranges $[l_p, u_p]$ und $[l_q, u_q]$ wenn:
$p + l_p > q + u_q$ oder $q + l_q > p + u_p$.

- DawnCC: conditional directives

→ Verringert data dependences → mehr Parallelisierung

DawnCC Beispiel: saxpy

```
1 void saxpy(float a, float* x, float* y, int n) {
2   for (int i = 0; i < n; ++i) {
3     y[i] = a * x[i] + y[i];
4   }
5 }
```

DawnCC Beispiel: saxpy

```
1  void saxpy(float a, float* x, float* y, int n) {
2    long long int tmp;
3    tmp = n - 1;
4    char x_y_alias_free = ((x > y + tmp) ||
5                           (y > x + tmp));
6    #pragma omp target data \
7                map(to:x[0:tmp+1]) map(tofrom:y[0:tmp+1]) \
8                if(x_y_alias_free)
9    #pragma omp parallel for if(x_y_alias_free)
10   for (int i = 0; i < n; ++i) {
11     y[i] = a * x[i] + y[i];
12   }
13 }
```

Coalescing

```
1  #pragma omp target data map(tofrom: x[0:n-1])
2  #pragma parallel for
3  for (int i = 0; i < n; i++) {
4      x[i] += i;
5  }
6
7  #pragma omp target data map(tofrom: x[0:n-1])
8  #pragma parallel for
9  for (int i = 0; i < n; i++) {
10     x[i] *= 2;
11 }
```

Coalescing

```
#pragma omp target data map(tofrom: x[0:n-1])
{
    #pragma parallel for
    for (int i = 0; i < n; i++) {
        x[i] += i;
    }
    #pragma parallel for
    for (int i = 0; i < n; i++) {
        x[i] *= 2;
    }
}
```

DawnCC Vor- & Nachteile

+ Findet Parallelisierbarkeit die ein Mensch evtl. nicht findet
± *Coalescing*: Optimiert in hoher Komplexität
− Verringert Lesbarkeit & Maintainability
  - clutter
  - coalescing
− Verringert ggf. Performance
  - Viele kurze Loops
  - Alias checking

  → Schedule & move overhead
  - Benutzerdefinierte Datenstrukturen

→ als Teil größerer Pipeline im Backend

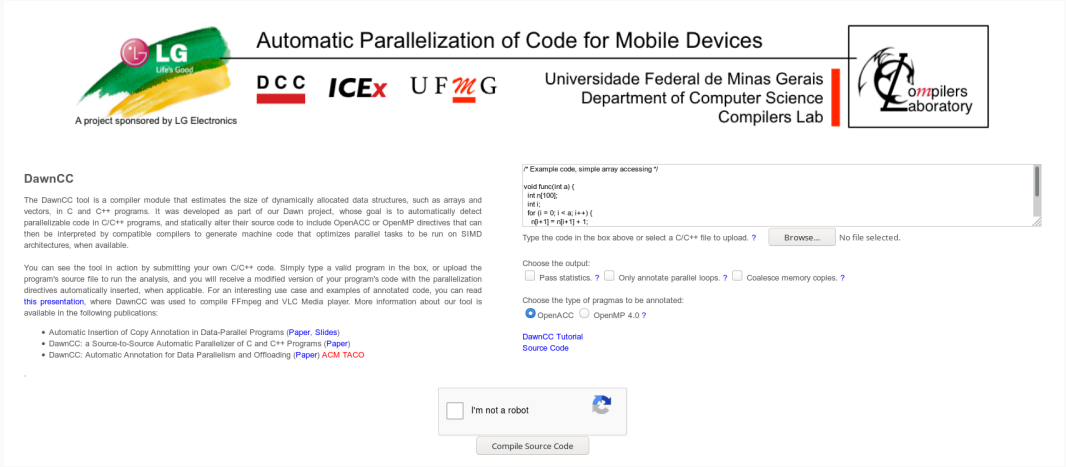
DawnCC Zusammenfassung

- Source-to-Source Compiler
- *Automatisiertes* Einfügen von Parallelisierungsdirektiven
  $\rightarrow$ Accelerator Offloading
- Erhöht Komplexität & Compiletime
- Verringert ggf. Laufzeit und Maintainability

$\rightarrow$ Nicht für jedes Programm angemessen

DawnCC Online Version

http://cuda.dcc.ufmg.br/dawn/index.php



Literatur (1)


Automatic Insertion of Copy Annotation in Data-Parallel Programs, G. Mendonca et al.
https://homepages.dcc.ufmg.br/~fernando/publications/papers/SBAC16_Gleison.pdf, Stand 10.11.2019


DawnCC: a Source-to-Source Automatic Parallelizer of C and C++ Programs,
B. Guimaraes et al.
https://homepages.dcc.ufmg.br/~fernando/publications/papers_pt/Breno16Tools.pdf, Stand 08.11.2019


DawnCC - Github repository.
https://github.com/gleisonsdm/DawnCC-Compiler, Stand 18.11.2019


Automatic Insertion of Copy Annotations in Data-Parallel Programs, G. Mendonca et al.
http://cuda.dcc.ufmg.br/dawn/include/papers/SBAC_PAD16.pdf, Stand 09.11.2019


The Program Dependence Graph and Its Use in Optimization, J. Ferrante et al. 1986.
https://cs.gmu.edu/~white/CS640/p319-ferrante.pdf, Stand 18.11.2019


CMPT886 part2: "Program Representations", 2015.
https://www.youtube.com/watch?v=5vvSJ_gWkCY, Stand 18.11.2019

Literatur (2)


The OpenACC Application Programming Interface.
https://www.openacc.org/sites/default/files/inline-files/OpenACC.2.7.pdf, Stand 18.11.2019


OpenMP Application Programming Interface.
https://www.openmp.org/wp-content/uploads/OpenMP-API-Specification-5.0.pdf, Stand 18.11.2019


Static single assignment form.
https://en.wikipedia.org/wiki/Static_single_assignment_form, Stand 18.11.2019


Control-flow graph.
https://en.wikipedia.org/wiki/Control-flow_graph, Stand 18.11.2019


LLVM Language Reference Manual.
http://llvm.org/docs/LangRef.html, Stand 18.11.2019


opt - LLVM optimizer.
http://llvm.org/docs/CommandGuide/opt.html, Stand 18.11.2019


Writing an LLVM Pass.
http://llvm.org/docs/WritingAnLLVMPass.html, Stand 18.11.2019

Symbolic Range Analysis: Beispiel IR

```
1  void test(int x) {
2      int y;
3      y = 3;
4      ++y;
5      if (x >= -5) {
6          if (x <= 2) {
7              y = y + x;
8              A[y]...
9          }
10     }
11 }
```

```
1  define void @test(i32) #0 {
2    %2 = alloca i32, align 4
3    %3 = alloca i32, align 4
4    store i32 5, i32* %3, align 4
5    %5 = add nsw i32 %4, 1
6    store i32 %5, i32* %3, align 4
7    %6 = load i32, i32* %2, align 4
8    %7 = icmp sge i32 %6, -5
9    br i1 %7, label %8, label %15
10
11 8:
12   %9 = load i32, i32* %2, align 4
13   %10 = icmp sle i32 %9, 5
14   br i1 %10, label %11, label %15
15
16 11:
17   %12 = load i32, i32* %2, align 4
18   %13 = load i32, i32* %3, align 4
19   %14 = add nsw i32 %13, %12
20   store i32 %14, i32* %3, align 4
21   br label %15
22
23 15:
24   ret void
25 }
```

DawnCC Beispiel: saxpy

```
1  void saxpy(float a, float* x, float* y, int n) {
2    long long int AI1[6];
3    AI1[0] = n + -1;
4    AI1[1] = 4 * AI1[0];
5    AI1[2] = AI1[1] + 4;
6    AI1[3] = AI1[2] / 4;
7    AI1[4] = (AI1[3] > 0);
8    AI1[5] = (AI1[4] ? AI1[3] : 0);
9    char RST_AI1 = 0;
10   RST_AI1 |= !(((void*) (x + 0) > (void*) (y + AI1[5]))
11   || ((void*) (y + 0) > (void*) (x + AI1[5])));
12   #pragma omp target data map(to: x[0:AI1[5]]) (tofrom: y[0:AI1[5]]) if(!RST_AI1)
13   #pragma omp target if(!RST_AI1)
14   #pragma omp parallel for
15   for (int i = 0; i < n; ++i) {
16     y[i] = a * x[i] + y[i];
17   }
18 }
```