GraphIt to CUDA compiler in 2021 LOC: A case for high-performance DSL implementation via staging with BuildDSL

Ajay Brahmakshatriya
Saman Amarasinghe
CSAIL, MIT

4th April 2022
DSLs enable high-performance

- Performance critical domains require *domain-specific* optimizations
DSLs enable high-performance

- Image Processing
- Graph Processing
- Sparse Array Processing

- Built for domain-experts with domain-experts!
DSLs enable high-performance

• Built for domain-experts with domain-experts!

• Built for domain-experts by domain-experts!

BuildIt!

Writing DSLs is hard

Typical DSL development cycle

Step 1
Design DSL abstractions

Step 2
Implement parser + IR

Step 3
Implement analyses and transforms

Step 4
Device specific code-generation

Requires compiler expertise

• Domain experts can and do write high-performance libraries
BuildIt

Multi-stage programming in C++

• BuildIt brings multi-stage programming to C++ in a lightweight way
BuildIt
Multi-stage programming in C++

- BuildIt brings multi-stage programming to C++ in a lightweight way
BuildIt

Multi-stage programming in C++

• BuildIt uses types to differentiate stages/binding times

\[ \text{dyn\_var}\langle T \rangle \quad \text{static\_var}\langle T \rangle \]
BuildIt
Multi-stage programming in C++

- BuildIt uses types to differentiate stages/binding times

\[ \text{dyn\_var}\langle T \rangle \]

Evaluation of expressions of type \text{dyn\_var}\langle T \rangle is delayed to next stage

\[
\begin{align*}
\text{dyn<int>} & \ x = 0; \\
\text{dyn<float>} & \ y = 3.14f; \\
& \ x = y + 2.71f; \\
\text{if} \ (x > 8.0f) \\
& \ \ \ \ y = 0; \\
\text{else} \\
& \ \ \ \ y = 1;
\end{align*}
\]

\[
\begin{align*}
\text{int} & \ x = 0; \\
\text{float} & \ = 3.14f; \\
& \ x = y + 2.71f; \\
\text{if} \ (x > 8.0f) \\
& \ \ \ \ y = 0; \\
\text{else} \\
& \ \ \ \ y = 1;
\end{align*}
\]
BuildIt

Multi-stage programming in C++

- BuildIt uses types to differentiate stages/binding times

\[
dyn\_var\langle T \rangle
\]
\[
static\_var\langle T \rangle
\]

- static\_var\langle T \rangle is completely evaluated in the current stage

```c++
static\_int \ x \ = \ 0;
dyn\_float \ y \ = \ 0.0f;
y \ = \ y \ + \ x;
if \ (x \ > \ 1)
    y \ = \ 0;
else
    y \ = \ 1;
```

```c++
float \ = \ 3.14f;
y \ = \ y \ + \ 0; \ // \ x \ = \ 0
    // no condition on x
y \ = \ 1;
```
BuildIt
Specialization with static_var(s)

dyn_var<int> power_f(dyn_var<int> base, 
       static_var<int> exponent) {
    dyn_var<int> res = 1, x = base;
    while (exponent > 1) {
        if (exponent % 2 == 1)
            res = res * x;
        x = x * x;
        exponent = exponent / 2;
    }
    return res * x;
}

int power_5(int arg0) {
    int var0 = arg0;
    int var1 = 1;
    int var2 = var0;
    var1 = var1 * var2;
    var2 = var2 * var2;
    var2 = var2 * var2;
    int var3 = var1 * var2;
    return var3;
}

exponent = 5
BuildIt
Specialization with static_var(s)

dyn_var<int> power_f(dyn_var<int> base,
    static_var<int> exponent) {
    dyn_var<int> res = 1, x = base;
    while (exponent > 1) {
        if (exponent % 2 == 1)
            res = res * x;
        x = x * x;
        exponent = exponent / 2;
    }
    return res * x;
}

int power_5 (int arg0) {
    int var0 = arg0;
    int var1 = 1;
    int var2 = var0;
    var1 = var1 * var2;
    var2 = var2 * var2;
    var2 = var2 * var2;
    int var3 = var1 * var2;
    return var3;
}

int power_10 (int arg0) {
    int var0 = arg0;
    int var1 = 1;
    int var2 = var0;
    var2 = var2 * var2;
    var1 = var1 * var2;
    var2 = var2 * var2;
    var2 = var2 * var2;
    int var3 = var1 * var2;
    return var3;
}

exponent = 10
BuildIt
Specialization with static_var(s)

dyn_var<int> power_f(dyn_var<int> base,
    static_var<int> exponent) {

dyn_var<int> res = 1, x = base;

    while (exponent > 1) {
        if (exponent % 2 == 1)
            res = res * x;
        x = x * x;
        exponent = exponent / 2;
    }

    return res * x;
}

int power_15 (int arg0) {
    int var0 = arg0;
    int var1 = 1;
    int var2 = var0;
    var1 = var1 * var2;
    var2 = var2 * var2;
    var1 = var1 * var2;
    var2 = var2 * var2;
    var1 = var1 * var2;
    var2 = var2 * var2;
    int var3 = var1 * var2;
    return var3;
}

exponent = 15
Staging libraries with BuildIt

A simple intuition

Program using the library

Domain Specific Library

BuildIt multistage programming framework

for (int x = ...)

High-performance code

vertex v = ...
Staging libraries with BuildIt
A simple intuition

Domain Specific Library
BuildIt multi-stage programming framework

 DSL input

 DSL Compiler

 High-performance code

 vertex v = ...

 for (int x = ...

 Domain Specific Analysis?
 Domain Specific Transformations?
 Scheduling?
 Device specific CPU/GPU/... code generation?
Building a DSL demo
Language for Einsum expressions

• Implement a DSL for Einsum expressions

\[ A[j] = B[j][k] \times C[k] \]

• Scheduling and parallelize for CPUs and GPUs

• Implement constant propagation analysis + transformation
Building a DSL demo
Language for Einsum expressions

\[ A[j] = B[j][k] \times C[k] \]

shorthand for

\[ \forall j, A[j] = \sum_{k=0}^{N} B[j][k] \times C[k] \]

- Implement loop nests for LHS and RHS
- Implement sum reduction
Building a DSL demo
Language for Einsum expressions

\[ \forall j, A[j] = \sum_{k=0}^{N} B[j][k] \times C[k] \]

```cpp
struct Tensor {
    float * data; std::vector<int> dims;
    operator = (T rhs_expr) {
        if (this->rank == 1) {
            for (int i = 0; i < this->dims[0]; i++) {
                data[i] = evaluate_rhs(rhs_expr, i);
            }
        } else if (this->rank == 2) {
            for (int i = 0; i < this->dims[0]; i++) {
                for (int j = 0; j < this->this[1]; j++) {
                    data[I * dim[0] + j] = evaluate_rhs(rhs_expr, i, j);
                }
            }
        } else if ...
    }
};
```
Building a DSL demo
Language for Einsum expressions

\[ A[j] = \sum_{k=0}^{N} B[j][k] \times C[k] \]

```cpp
struct Tensor {
    float * data; std::vector<int> dims;

    void recursive_loop(int idx, int offset, T rhs) {
        if (idx < dims.size()) {
            for (int i = 0; i < dims[idx]; i++) {
                recursive_loop(idx + 1, offset + ..., rhs(i));
            }
        } else {
            data[offset] = rhs;
        }
    }

    operator = (T rhs_expr) {
        recursive_loop(0, rhs_expr);
    }
};
```
Building a DSL demo
Language for Einsum expressions

∀ j, A[j] = \sum_{k=0}^{N} B[j][k] * C[k]

```
struct Tensor {
  float * data; std::vector<int> dims;

  void recursive_loop(int idx, int offset, T rhs) {
    if (idx < dims.size()) {
      for (int i = 0; i < dims[idx]; i++) {
        recursive_loop(idx + 1, offset + ... , rhs(i));
      }
    } else {
      data[offset] = rhs;
    }
  }

  operator=(T rhs_expr) {
    recursive_loop(0, rhs_expr);
  }
};
```

Same approach for recursively iterating over RHS free variables!
Building a DSL demo

Source code

Demo source code

https://github.com/BuildIt-lang/einsum-lang

BuildIt

https://buildit.so
GraphIt DSL

• A real world state of the art graph DSL

2. Brahmakshatriya et al., Compiling Graph Algorithms for GPUs with GraphIt. International Symposium on Code Generation and Optimization (CGO 2021)
GraphIt DSL on GPUs

State of the art performance

<table>
<thead>
<tr>
<th>Optimization</th>
<th>Gunrock</th>
<th>GSWITCH</th>
<th>SEP-Graph</th>
<th>G2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load Balancing</td>
<td>VERTEX BASED, EDGE BASED, TWC</td>
<td>CM, WM, TWC, STRICT</td>
<td>VERTEX BASED</td>
<td>ETWC, TWC, STRICT, CM, WM, VERTEX BASED, EDGE BASED</td>
</tr>
<tr>
<td>Edge Blocking</td>
<td>Not supported</td>
<td>Not supported</td>
<td>Not supported</td>
<td>Supported</td>
</tr>
<tr>
<td>Vertex Set Creation</td>
<td>Fused/Unfused</td>
<td>Fused/Unfused</td>
<td>Fused</td>
<td>Span Tree/ Bitmap/Boomer</td>
</tr>
<tr>
<td>Direction Optimization</td>
<td>Push/Pull/Hybrid</td>
<td>Push/Pull/Hybrid</td>
<td>Push/Pull/Hybrid</td>
<td>Push/Pull/Hybrid</td>
</tr>
<tr>
<td>Deduplication</td>
<td>Supported</td>
<td>Not supported</td>
<td>Supported</td>
<td>Supported</td>
</tr>
<tr>
<td>Vertex Ordering</td>
<td>Supported</td>
<td>Supported</td>
<td>Supported</td>
<td>Supported</td>
</tr>
<tr>
<td>Kernel Fusion</td>
<td>Supported</td>
<td>Not supported</td>
<td>Supported</td>
<td>Supported</td>
</tr>
<tr>
<td>Total combinations</td>
<td>48</td>
<td>32</td>
<td>16</td>
<td>576</td>
</tr>
</tbody>
</table>

Largest Scheduling Space

State of the art performance

23,783 lines of compiler code!!!
GraphIt DSL with BuildIt

Obtain comparable performance with GraphIt compiler for 45 experiments

In just 2021 lines of code on top of BuildIt!
BuildIt
Write high-performance DSLs with ease

BuildIt is available open source under the MIT license at https://buildit.so

Try Online: buildit.so/tryit/

Ajay Brahmakshatriya (ajaybr@mit.edu)