Single Assignment C:
A High Productivity Language for High Performance Computing

Clemens Grelck

Universiteit van Amsterdam

Workshop
Trends in High-Performance Distributed Computing
Amsterdam, Netherlands
March 14, 2012
The HP^3 Triangle in High Performance Computing

High Productivity  High Performance

High Portability
HPC in the Age of Multi- and Many-Core

Welcome to the Cluster Node Hardware Zoo!!
Welcome to the Cluster Node Hardware Zoo!!
HPC in the Age of Multi- and Many-Core

Welcome to the Cluster Node Hardware Zoo!!

Clemens Grelck, Universiteit van Amsterdam
HPC in the Age of Multi- and Many-Core

Welcome to the Cluster Node Hardware Zoo!!
HPC in the Age of Multi- and Many-Core

Welcome to the Cluster Node Hardware Zoo!!
HPC in the Age of Multi- and Many-Core

Welcome to the Cluster Node Hardware Zoo!!

The Future
Brought to You by AMD
Introducing the AMD APU Family
HPC in the Age of Multi- and Many-Core

Welcome to the Cluster Node Hardware Zoo!!

The Future
Brought to You by AMD
Introducing the AMD APU Family

Clemens Grelck, Universiteit van Amsterdam
HPC in the Age of Multi- and Many-Core

Compute node hardware is a zoo:

- Vastly different numbers of cores
- Vastly different core architectures: general, restricted
- Vastly different memory architectures
Compute node hardware is a zoo:

- Vastly different numbers of cores
- Vastly different core architectures: general, restricted
- Vastly different memory architectures

Programming diverse hardware is uneconomic:

- Various low-level programming models
- Each requires different expert knowledge
- Heterogeneous combinations of the above?
- Cumbersome, error-prone and inefficient
An Alternative: Single Assignment C (SAC)

Credo: abstraction, abstraction, abstraction

► Program what to compute, not exactly how
► Leave concrete organisation of execution to compiler and runtime system
► Put expert knowledge into tools, not into applications
► Architecture-agnostic programming for portability
► Compile one source to diverse target hardware
► Automatically manage resources: memory, cores, etc
SAC — Design Space

High-level functional, data-parallel programming with vectors, matrices, arrays

Easy to adopt for programmers with an imperative background

Suitability to achieve high performance in sequential and parallel execution

Clemens Grelck, Universiteit van Amsterdam
SAC — Design Space

High-level functional, data-parallel programming with vectors, matrices, arrays

SAC
High-level functional, data-parallel programming with vectors, matrices, arrays

SAC

Suitability to achieve high performance in sequential and parallel execution
SAC — Design Space

High-level functional, data-parallel programming with vectors, matrices, arrays

SAC

Easy to adopt for programmers with an imperative background

Suitability to achieve high performance in sequential and parallel execution
Case Study: 1-Dimensional Complex FFT (NAS-FT)

```
complex[.] FFT(complex[.] v, complex[.] rofu) {
    even = condense(2, v);
    odd = condense(2, drop([1], v));

    even = FFT(even, rofu);
    odd = FFT(odd, rofu);

    rofu = condense(len(rofu) / len(odd), rofu);

    left = even + odd * rofu;
    right = even - odd * rofu;

    return left ++ right;
}
```
Case Study: 1-Dimensional Complex FFT (NAS-FT)

Now, what is functional array programming?
Functional Array Programming

```c
complex[.] FFT(complex[.] v, complex[.] rofu)
{
    even = condense(2, v);
    odd = condense(2, drop([1], v));
    even = FFT(even, rofu);
    odd = FFT(odd, rofu);
    rofu = condense(len(rofu) / len(odd), rofu);
    left = even + odd * rofu;
    right = even - odd * rofu;
    return left ++ right;
}
```

Role of Functions:

- Map argument values to result values
- No side effects
- Call-by-value parameter passing
Functional Array Programming

```c
complex[.] FFT(complex[.] v, complex[.] rofu) {
    even = condense(2, v);
    odd = condense(2, drop([1], v));
    even = FFT(even, rofu);
    odd = FFT(odd, rofu);
    rofu = condense(len(rofu) / len(odd), rofu);
    left = even + odd * rofu;
    right = even - odd * rofu;
    return left ++ right;
}
```

Role of Variables:
- Variables are placeholders for values
- Variables do not denote memory locations
- Automatic memory management
Functional Array Programming

```
complex[.] FFT(complex[.] v, complex[.] rofu)
{
    even = condense(2, v);
    odd = condense(2, drop([1], v));

    even = FFT( even, rofu);
    odd = FFT( odd, rofu);

    rofu = condense( len( rofu ) / len( odd ), rofu );

    left = even + odd * rofu;
    right = even - odd * rofu;

    return left ++ right;
}
```

Execution Model:

- Contextfree substitution of expressions
- No side-effects
Functional Array Programming

```c
complex[.] FFT(complex[.] v, complex[.] rofu) {
    even = condense(2, v);
    odd = condense(2, drop([1], v));
    even = FFT(even, rofu);
    odd = FFT(odd, rofu);
    rofu = condense(len(rofu) / len(odd), rofu);
    left = even + odd * rofu;
    right = even - odd * rofu;
    return left ++ right;
}
```

Control flow constructs:

- Branches are syntactic sugar for conditional expressions
- Loops are syntactic sugar for tail-end recursive functions
- Data flow determines execution order
Functional Array Programming

```c
complex[.] FFT(complex[.] v, complex[.] rofu) {
    even = condense(2, v);
    odd = condense(2, drop([1], v));
    even = FFT(even, rofu);
    odd = FFT(odd, rofu);
    rofu = condense(len(rofu) / len(odd), rofu);
    left = even + odd * rofu;
    right = even - odd * rofu;
    return left ++ right;
}
```

Nature of Arrays:

- Pure values, mapping indices to (other) values
- No state, no fixed memory representation
- Maybe no memory manifestation at all
Case Study: 3-Dimensional Complex FFT (NAS-FT)

**Algorithmic idea:**

![Diagram showing the algorithmic idea of the 3-Dimensional Complex FFT](image)

**Implementation:**

```c
complex[.,.,.] FFT( complex[.,.,.] a, complex[.] rofu) {
    b = { [,y,z] -> FFT( a[.,y,z], rofu) };
    c = { [x,.,z] -> FFT( b[x,.,z], rofu) };
    d = { [x,y,.] -> FFT( c[x,y,], rofu) };

    return d;
}

typedef double [2] complex;
```

Clemens Grelck, Universiteit van Amsterdam
The Same in Fortran

```fortran
subroutine fft(dir, x1, x2)  
  implicit none  
  include 'global.h'  
  integer dir  
enddo
enddo

double complex scratch(fftblockpad, w, y)  

if (dir .eq. 1) then  
  call cffts1(is, d(1), x1, scratch)  
  endif
enddo

integer is, d(3), logd(3)  
enddo
enddo

double complex xout(i+ii, j, k) = y(i, j, 1)  
enddo
enddo

double complex (d(1), d(2), d(3))  
enddo
enddo

integer i, j, k, ii
enddo
enddo

integer i, j, k, ii
enddo
enddo

if (1 .eq. m) goto 160
endif
```

Clemens Grelck, Universiteit van Amsterdam

Single Assignment C
The Power of Abstraction

- Programming by composition of building blocks:
  - Rapid prototyping
  - High confidence in correctness
  - Good readability of code
  - Plenty of code reuse opportunities

- Opportunities for compiler and runtime system:
  - Target-independent optimisation
  - Code generation for variety of target architectures
  - Automatic parallelisation
  - Automatic memory management

- Result:
  - Sufficient performance
  - For a range of architectures
  - Without extra effort
Compilation Challenge

And achieve reasonable performance....

Goal: achieve 90% with 10% of the effort
SAC as a Compiler Technology Project

Some Figures:

▶ SAC compiler + runtime library:
  ▶ 300,000 lines of code, about 1000 files
  ▶ about 250 compiler passes
  ▶ + standard prelude + standard library
  ▶ More than 15 years of research and development
  ▶ More than 30 people involved over the years

Involved Universities:

▶ University of Kiel, Germany (1994–2005)
▶ University of Toronto, Canada (since 2000)
▶ University of Lübeck, Germany (2001–2008)
▶ University of Amsterdam, Netherlands (since 2008)
▶ Heriot-Watt University, Scotland (since 2011)
Conclusion

Single Assignment C:

- reconcile productivity, portability and performance
- one source for all architectures
- functional array programming
- exposure of fine-grained concurrency
- automatically sequentialising compiler
- automatic resource management

www.sac-home.org