# The DAO of Parallel Software Construction

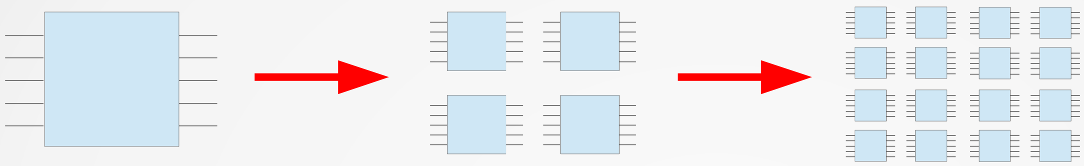
Armin Größlinger
Department of Informatics and Mathematics
University of Passau

ICSOFT 2013

Reykjavík, Iceland
2013-07-30

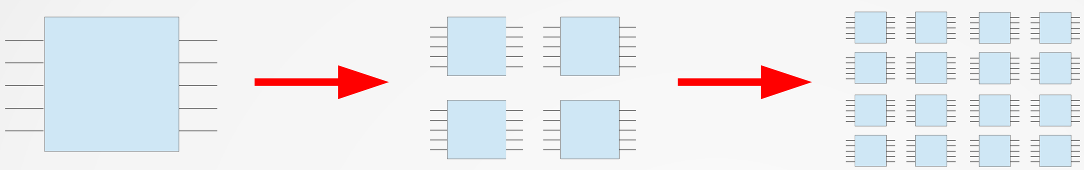
# The Need for Parallel Software

We are going from “Moores” to “Cores”

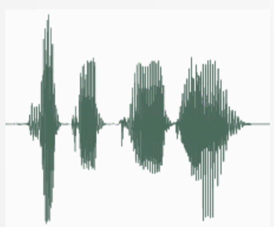


# The Need for Parallel Software

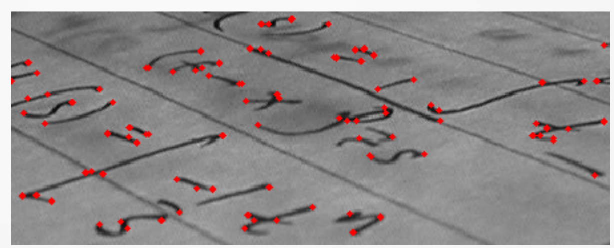
We are going from “Moores” to “Cores”



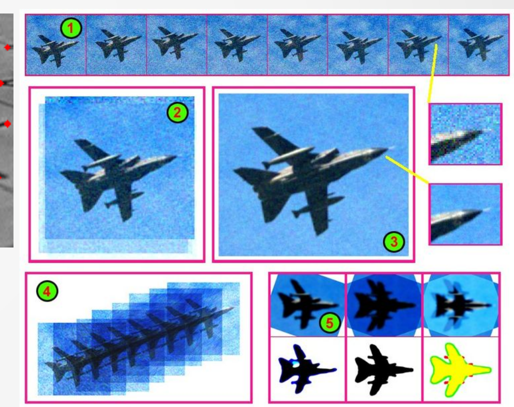
New applications:

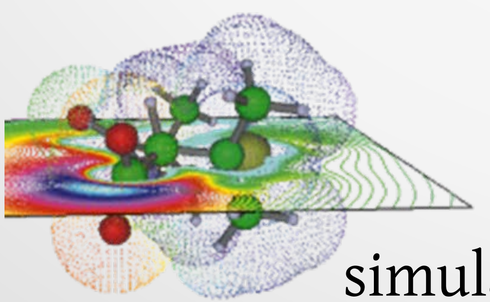


speech recognition



real-time image & video processing





simulation

online data analysis

# D A O

In this talk, I will argue to use a

- **D**omain-specific approach,
- **A**nalysis of the problem and domain parameters, and
- **O**ptimization using automated techniques

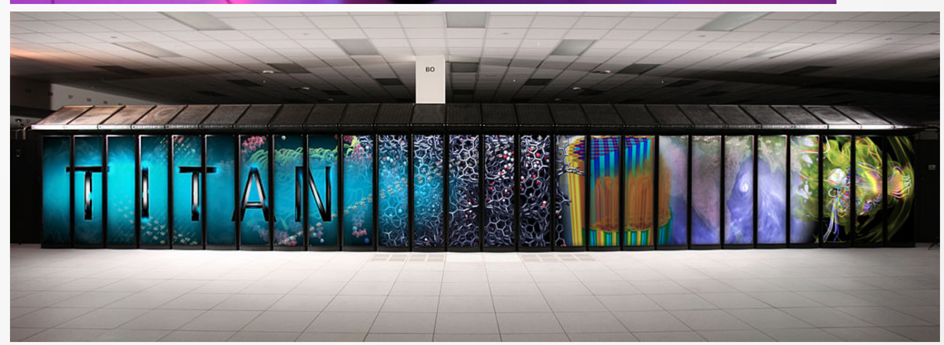
to construct parallel programs.

(In addition, “DAO” matches syntactically with the main concept of daoist philosophy; therefore, there are a few quotes on the slides, mainly from the *Dao De Jing* (DDJ).)

# Today's Supercomputers

Tianhe-2 (China)
3.120.000 Cores
33 PetaFLOPS
17.8 MW

Titan (US)
560.640 Cores
17.6 PetaFLOPS




JUQUEEN (Germany)
458.752 Cores
5 PetaFLOPS

# GPUs and  Accelerators

Graphics processors (GPUs) and dedicated accelerators

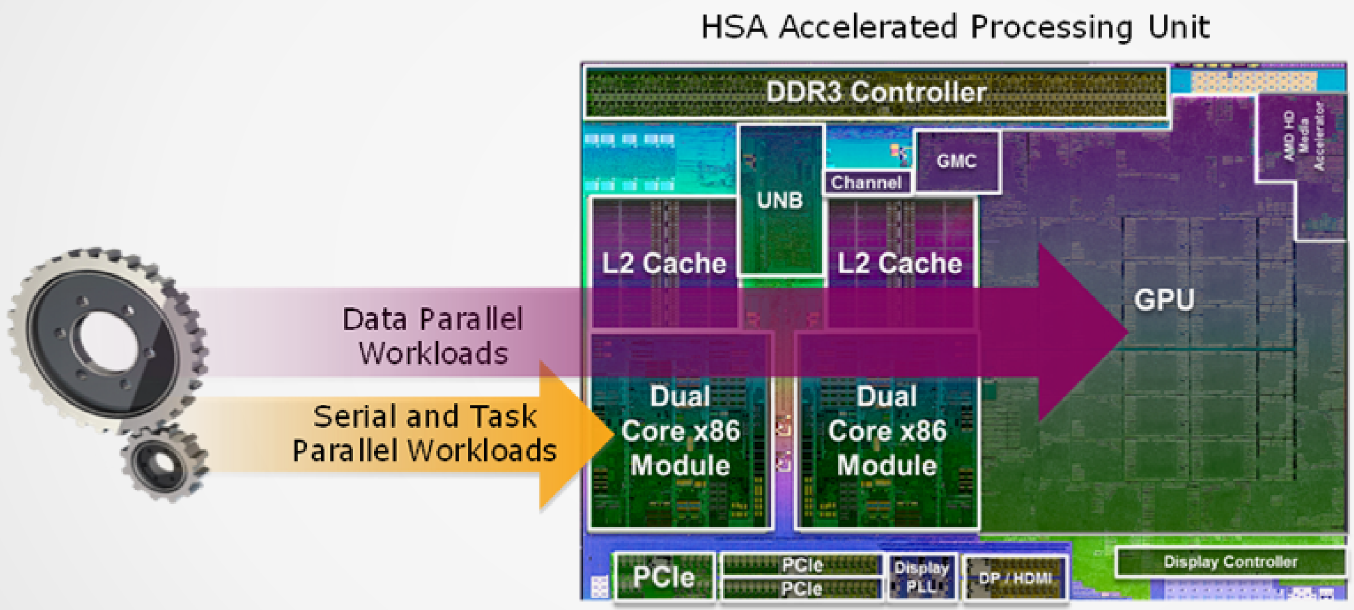
- deliver 1-10 TeraFLOPS for 100-10.000 $



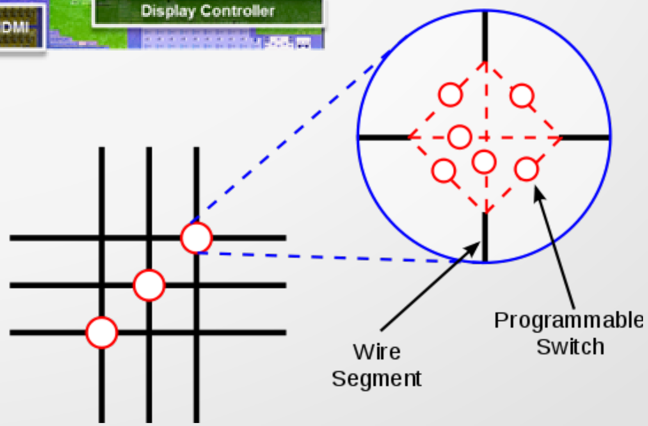
- achieve ≥20 GigaFLOPS per Watt

# Heterogeneous and Reconfigurable Hardware

- Heterogeneous hardware is becoming mainstream




- Even reconfigurable hardware



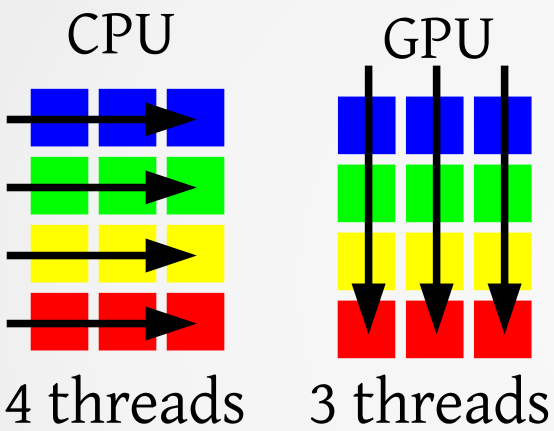
# Hardware Diversity

Performance is not portable from one architecture to another.

"The more you experience, the less you know." (DDJ, Sec. 47)

# Hardware Diversity

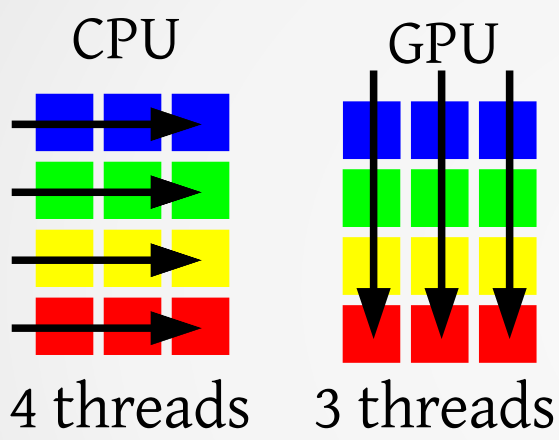
Performance is not portable from one architecture to another.




"The more you experience, the less you know." (DDJ, Sec. 47)

# Hardware Diversity

Performance is not portable from one architecture to another.

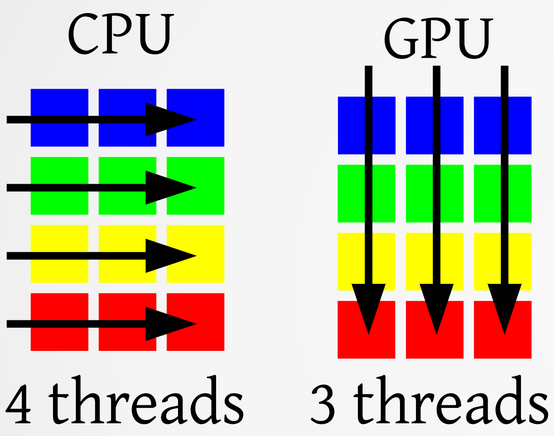



Complex rules for performance, e.g.:
Alignment is good for performance...

“The more you experience, the less you know.” (DDJ, Sec. 47)

# Hardware Diversity

Performance is not portable from one architecture to another.

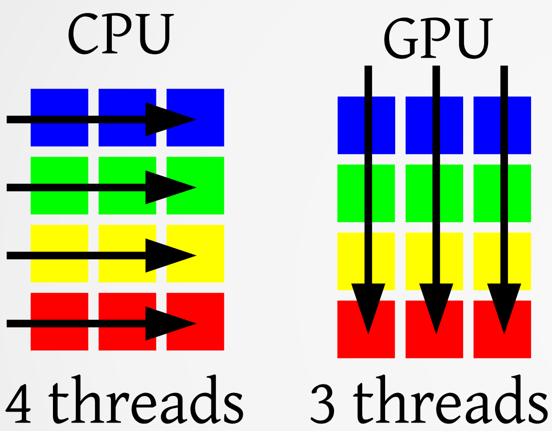



Complex rules for performance, e.g.:
Alignment is good for performance...
... except when mis-alignment is better.

“The more you experience, the less you know.” (DDJ, Sec. 47)

# Hardware Diversity

Performance is not portable from one architecture to another.




Complex rules for performance, e.g.:
Alignment is good for performance...
... except when mis-alignment is better.

Our experience: do not trust benchmarks
→ too many “random” effects on today's processors

“The more you experience, the less you know.” (DDJ, Sec. 47)

# Traditional Parallel Programming

- Hire a programmer/student/expert/... to hack on the parallel code.
- Many hours/days/weeks of work and performance experiments necessary.



“It is easier to lose a yard than take an inch.” (DDJ, Sec. 69)

# Traditional Parallel Programming

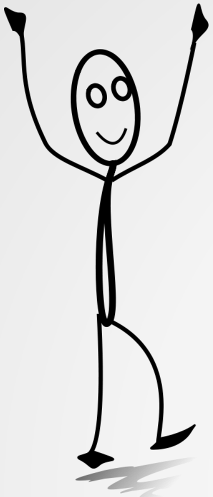
- Hire a programmer/student/expert/... to hack on the parallel code.
- Many hours/days/weeks of work and performance experiments necessary.
- Need to repeat for every new hardware platform.

...



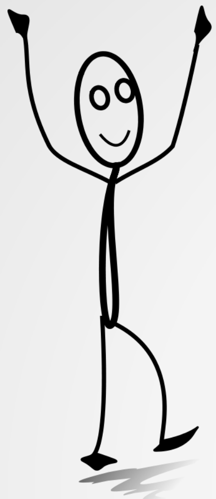
"It is easier to lose a yard than take an inch." (DDJ, Sec. 69)

# How to Make Users Happy



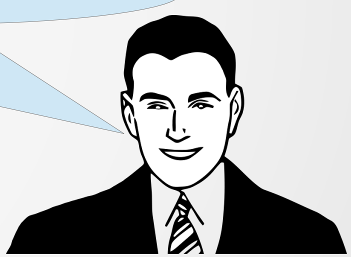
Reduce effort for users/programmers

# How to Make Users Happy

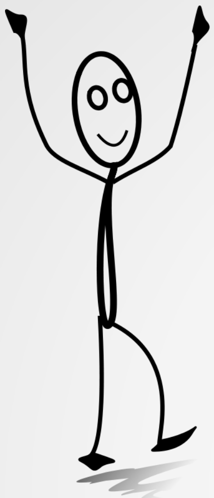


Reduce effort for users/programmers

“Progress in software engineering can only be achieved by abstraction“ (SE wisdom)

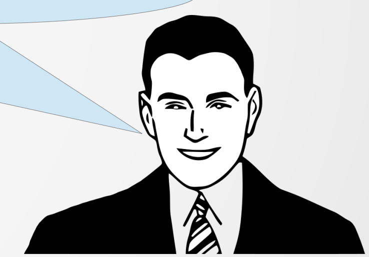


# How to Make Users Happy



Reduce effort for users/programmers

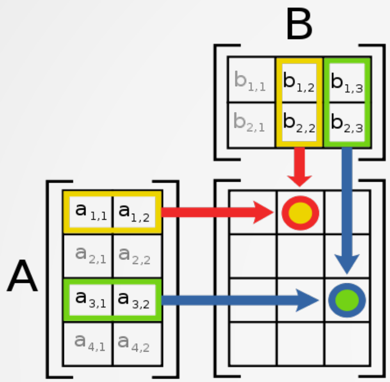
"Progress in software engineering can only be achieved by abstraction" (SE wisdom)



But: Abstraction and high performance do not mix a priori.

# Abstraction

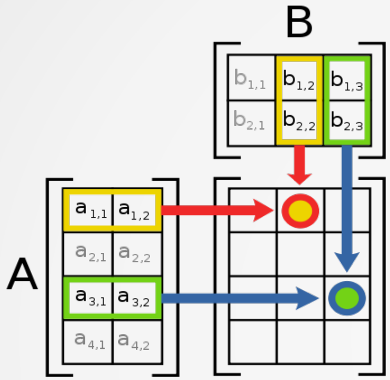
## Something simple: Matrix-Matrix-Multiply




Assume A and B are distributed row-wise in block-cyclical fashion. Which elements of A and B have to be sent over the network to compute A·B?

# Abstraction

Something simple: Matrix-Matrix-Multiply

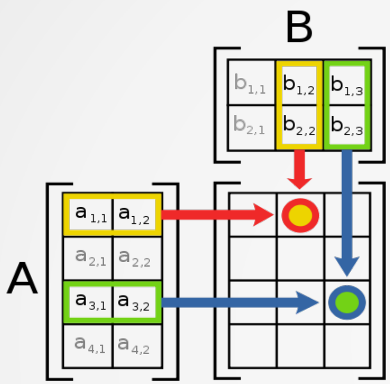



Assume A and B are distributed row-wise in block-cyclical fashion. Which elements of A and B have to be sent over the network to compute A·B?

Isn't this question quite ridiculous?

# Abstraction

Something simple: Matrix-Matrix-Multiply




Assume A and B are distributed row-wise in block-cyclical fashion. Which elements of A and B have to be sent over the network to compute A·B?

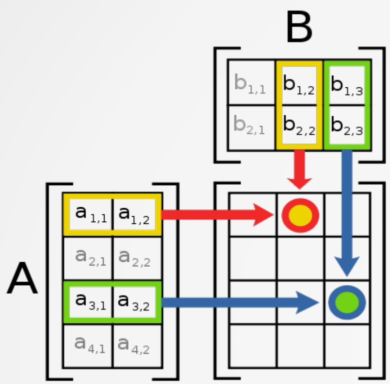
Isn't this question quite ridiculous?

We do not want to write

```
MPI_Datatype elems;
...
for (i=...) {
   for (j=...) {
      MPI_Recv(..., elems, ...);
      for (k=...)
         C[i][j] += ... ;
   }
}
```

# Abstraction

Something simple: Matrix-Matrix-Multiply




Assume A and B are distributed row-wise in block-cyclical fashion. Which elements of A and B have to be sent over the network to compute A·B?

Isn't this question quite ridiculous?

We do not want to write

```
MPI_Datatype elems;
...
for (i=...) {
   for (j=...) {
      MPI_Recv(..., elems, ...);
      for (k=...)
         C[i][j] += ... ;
   }
}
```

We want to write

C = A*B;

# Knowledge for Optimization

- “C=A*B” is possible in High-Performance Fortran (HPF), but HPF was successful in a niche only.
- Compiler needs more information for aggressive optimization.

# Knowledge for Optimization

- “C=A*B” is possible in High-Performance Fortran (HPF), but HPF was successful in a niche only.
- Compiler needs more information for aggressive optimization.
- Make the knowledge explicit!
- Are you writing similar codes again and again?
  → Don't waste your time hand-optimizing code in a general purpose language, use a simple language tailored to the application problem!

# Domain-specific Approach

- Design a domain-specific language (DSL).
- Restrict to the required language constructs only.
- DSLs excludes situations bad for the optimizer *a priori*, e.g.
  - no aliasing
  - no irregular arrays
  - no pointer arithmetic, often no pointers at all
  - no statements with side-effects

"The follower of the DAO forgets as much as he can every day." (DDJ, Sec. 48)

# Tool: Spiral

- Generator for linear transforms (DFT, DCT, etc.)
- Uses several DSLs to transform a specification into efficient code:
  - start with a specification, e.g. $\mathrm{DFT}_n$ for a DFT of a particular size $n$
  - apply rules which transform the specification step-by-step
- Beats other implementations (libraries and generated codes) for linear transforms.

M. Püschel, F. Franchetti and Y. Voronenko. *Spiral.* In D. Padua et al., eds., Encyclopedia of Parallel Computing, Springer-Verlag, September 2011

# DSLs in Spiral I

- Rewrite system for algebraic expressions

$$
\begin{aligned}
\mathrm{DFT}_n &\rightarrow (\mathrm{DFT}_k \otimes I_m)\, T^n_m\, (I_k \otimes \mathrm{DFT}_m)\, L^n_k, \\
\mathrm{DFT}_n &\rightarrow V_n^{-1} (\mathrm{DFT}_k \otimes I_m)(I_k \otimes \mathrm{DFT}_m) V_n, \\
\mathrm{DFT}_n &\rightarrow W_n^{-1} (I_1 \oplus \mathrm{DFT}_{p-1}) E_n (I_1 \oplus \mathrm{DFT}_{p-1}) W_n, \\
\mathrm{DFT}_n &\rightarrow B'_n D_m\, \mathrm{DFT}_m\, D'_m\, \mathrm{DFT}_m\, D''_m B_n, \\
\mathrm{DFT}_n &\rightarrow P^T \ldots (\mathrm{DFT}_{2m} \oplus (I_{k-1} \otimes_i C_{2m}\, \mathrm{rDFT}_{2m}(i/2k)))\, (\mathrm{RDFT}_{2k} \otimes I_m),
\end{aligned}
$$

- Rewrite system to generate loops

| SPL expression $S$ | Pseudo code for $y = Sx$ |
|---|---|
| $A_n B_n$ | <code for: t = Bx><br><code for: y = At> |
| $I_m \otimes A_n$ | for (i=0; i<m; i++)<br><code for: y[i*n:1:i*n+n-1] = A(x[i*n:1:i*n+n-1])> |

# DSLs in Spiral II

- Rewrite system for parallelism

$$\underbrace{AB}_{\text{smp}(p,\mu)} \rightarrow \underbrace{A}_{\text{smp}(p,\mu)} \underbrace{B}_{\text{smp}(p,\mu)}$$

$$\underbrace{A_m \otimes I_n}_{\text{smp}(p,\mu)} \rightarrow \underbrace{\left(L^{mp}_m \otimes I_{n/p}\right)\left(I_p \otimes \left(A_m \otimes I_{n/p}\right)\right)\left(L^{mp}_p \otimes I_{n/p}\right)}_{\text{smp}(p,\mu)}$$

$$\underbrace{L^{mn}_m}_{\text{smp}(p,\mu)} \rightarrow \begin{cases} \underbrace{\left(I_p \otimes L^{mn/p}_{m/p}\right)}_{\text{smp}(p,\mu)} \underbrace{\left(L^{pn}_p \otimes I_{m/p}\right)}_{\text{smp}(p,\mu)} \\ \underbrace{\left(L^{pm}_m \otimes I_{n/p}\right)}_{\text{smp}(p,\mu)} \underbrace{\left(I_p \otimes L^{mn/p}_m\right)}_{\text{smp}(p,\mu)} \end{cases}$$

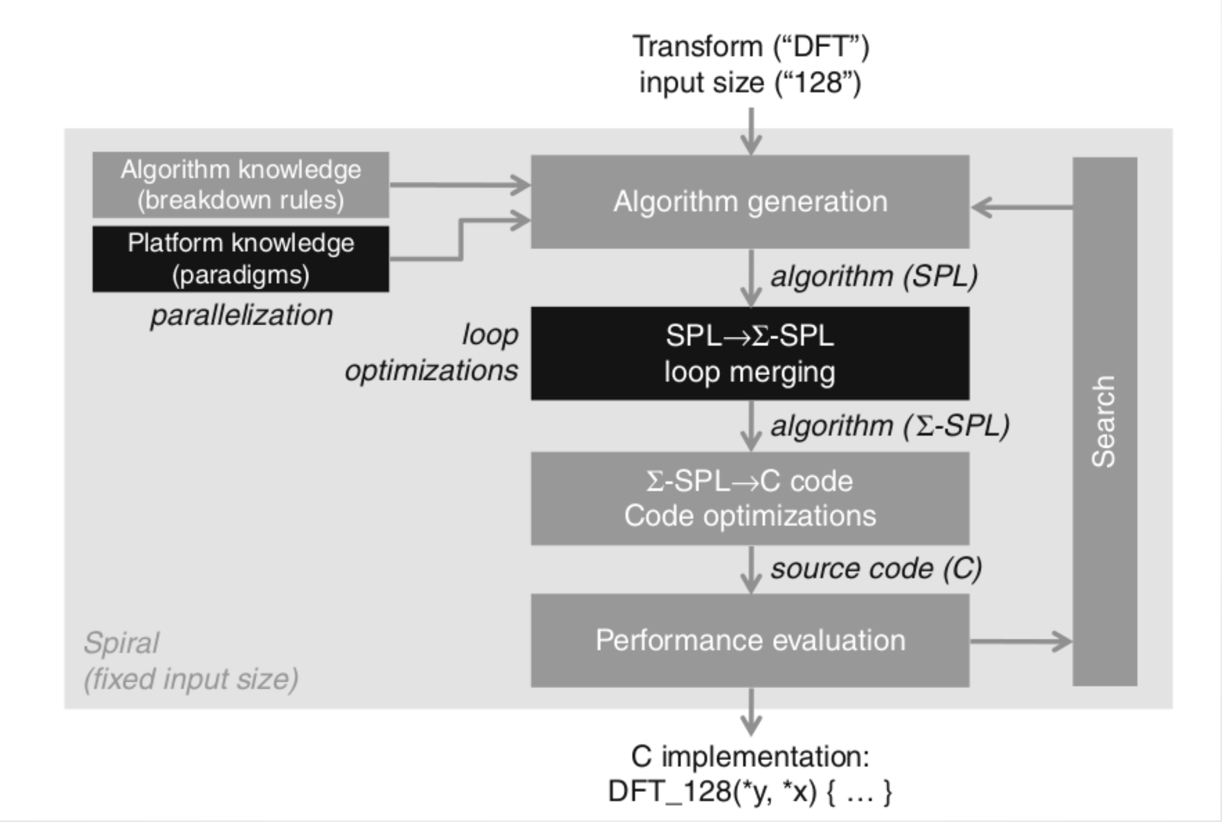
$$\underbrace{I_m \otimes A_n}_{\text{smp}(p,\mu)} \rightarrow I_p \otimes \left(I_{m/p} \otimes A_n\right)$$

$$\underbrace{\left(P \otimes I_n\right)}_{\text{smp}(p,\mu)} \rightarrow \left(P \otimes I_{n/\mu}\right) \otimes I_\mu$$

$p$: number of processors, $\mu$: cache line size

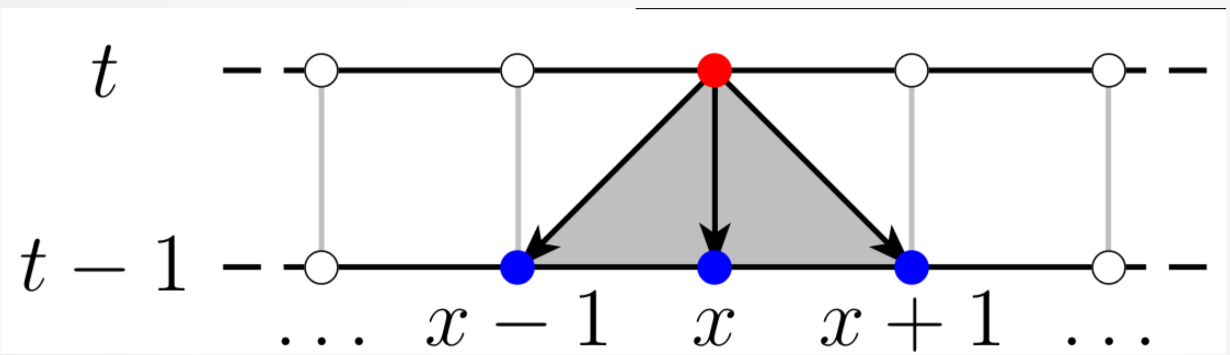
# Spiral Big Picture

- Rewrite engines combined with machine learning
- Platform characteristics ("paradigms") present in rewrite rules



# Tool: Pochoir

- Compiler for stencil computations
- DSL embedded in C++
- Example:

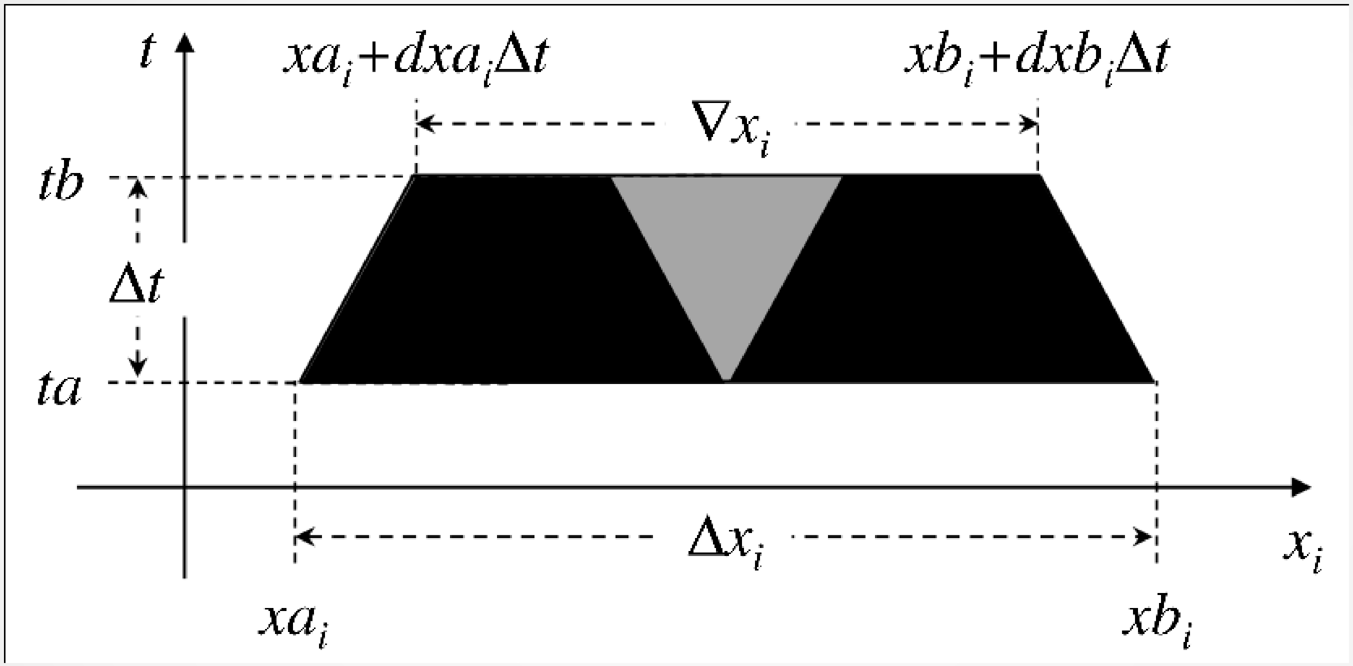



$$U_t(x) = U_{t-1}(x-1) - 2 \cdot U_{t-1}(x) + U_{t-1}(x+1)$$

Y. Tang, R. Chowdhury, C. Luk, B. Kuszmaul, C. Leiserson, *The Pochoir Stencil Compiler*. SPAA'11

# Pochoir: Parallelization

- Main idea: "hyperspace cut" (applied recursively)




- Split iteration domain in
  - pieces not requiring communication (black)
  - pieces having to wait for other data (grey)
- Execute black pieces first, then grey pieces.

# Tool: Halide

- DSL embedded into C++ for image processing
- Main characteristic: separation of algorithm and schedule
  - algorithm: functional description of computation
  - schedule: execution order of operations and storage locations for computed values

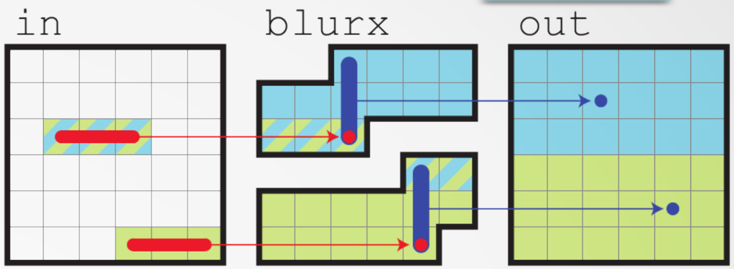
J. Ragan-Kelley, C. Barnes, A. Adams, S. Paris, F. Durand, S. Amarasinghe. *Halide: A Language and Compiler for Optimizing Parallelism, Locality, and Recomputation in Image Processing Pipelines*. PLDI 2013

# Halide Example

- Algorithm

```
UniformImage in(UInt(8),2);
Var x, y;
Func blurx(x,y) = (in(x-1,y) + in(x,y) + in(x+1,y))/3;
Func out(x,y)   = (blurx(x,y-1) + blurx(x,y) + blurx(x,y+1))/3;
```

# Halide Example




- Algorithm

```
UniformImage in(UInt(8),2);
Var x, y;
Func blurx(x,y) = (in(x-1,y) + in(x,y) + in(x+1,y))/3;
Func out(x,y)   = (blurx(x,y-1) + blurx(x,y) + blurx(x,y+1))/3;
```

- Schedule

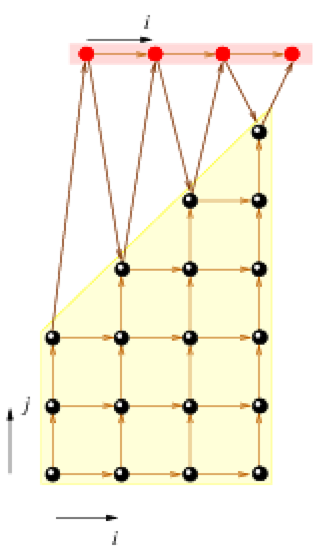
```
out.tile(x, y, xi, yi, 256, 32).vectorize(xi,8).parallel(y);
blurx.chunk(x).vectorize(x,8);
```

- Finding a schedule:
  - few degrees of freedom: “tile”, “vectorize”, etc.
  - can be specified by user
  - auto-tuning using genetic algorithm

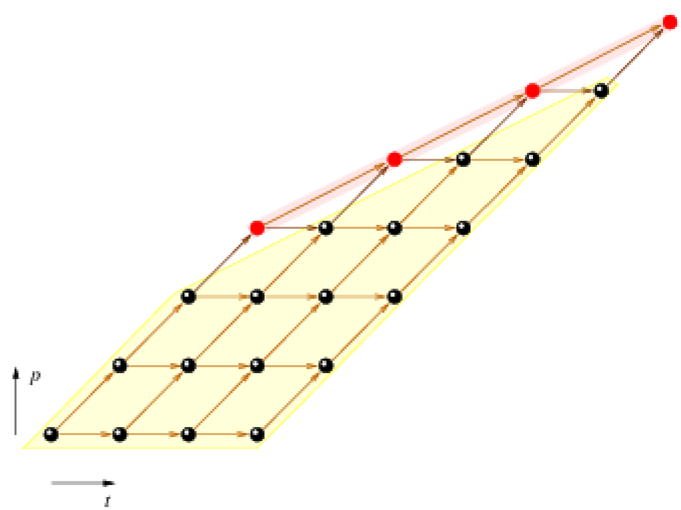
# Technique: Polyhedral Compilation

for $i = 1$ to $n$ do
  for $j = 0$ to $i + m$ do
    $A(i,j) = A(i-1,j) + A(i,j-1)$
  od
  $A(i,i+m+1) = A(i-1,i+m) + A(i,i+m)$
od

for $t = 0$ to $m+2*n-1$ do
  parfor $p = \max(0, t-n+1)$ to $\min(t, \lceil (t+m)/2 \rceil)$ do
    if $2*p = t+m+1$ then
  $S_2$ : $A(p-m, p+1) = A(p-m-1, p) + A(p-m, p)$
    else
  $S_1$ : $A(t-p+1, p+1) = A(t-p, p+1) + A(t-p+1, p)$
    fi
  od
od




Source dependence graph




Target dependence graph

P. Feautrier and C. Lengauer. *Polyhedron Model*. In D. Padua et al., eds., Encyclopedia of Parallel Computing, Springer-Verlag, September 2011

# Polyhedral Compilation

- Developed since 1980s, roots go back to late 1960s.
- Power comes from the use of linear algebra and integer linear programming.
- Not a DSL but polyhedral representation has powerful laws for program transformation.
- Slowly comes out from its niche into the “real” world.

A. Simbürger, S. Apel, A. Größlinger, C. Lengauer. *The Potential of Polyhedral Optimization: An Empirical Study*. Automated Software Engineering 2013, to appear

# Why are the Tools/Techniques Successful?

- They are **Domain**-specific:
  - domain is narrow enough to have powerful laws (algebraic properties)
  - domain is broad enough: not every interesting code has been or will be written by hand
  - domain is well understood and has many applications

"Let your community be small, with only a few people" (DDJ, Sec. 80)

# Why are the Tools/Techniques Shown Successful?

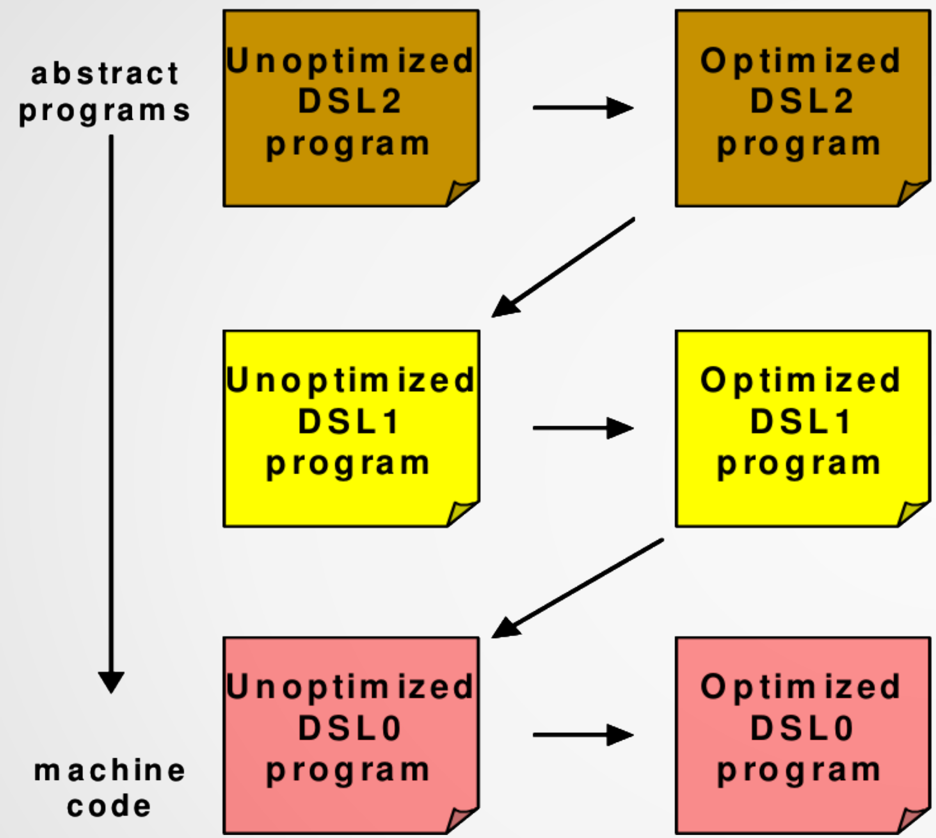
- **Analysis** of the domain:
  - Know the laws of the domain
  - Know (almost) all the factors that influence performance
- **Analysis** of programs in the domain:
  - Compiler can extract required knowledge for optimization
  - Factors influencing performance are turned into parameters for an optimization problem
  - Automatically discriminate between correct and incorrect choices for the parameters

# Why are the Tools/Techniques Shown Successful?

- **Optimization**
  - analytical optimization over several levels
    - rewrite systems
    - optimization w.r.t. an objective function
  - select parameters through
    - auto-tuning (e.g., genetic algorithms, sampling)
    - machine learning

"[one of the three treasures is] restraint, by which one finds strength" (DDJ, Sec. 67)

# Hierarchical DSL Optimization

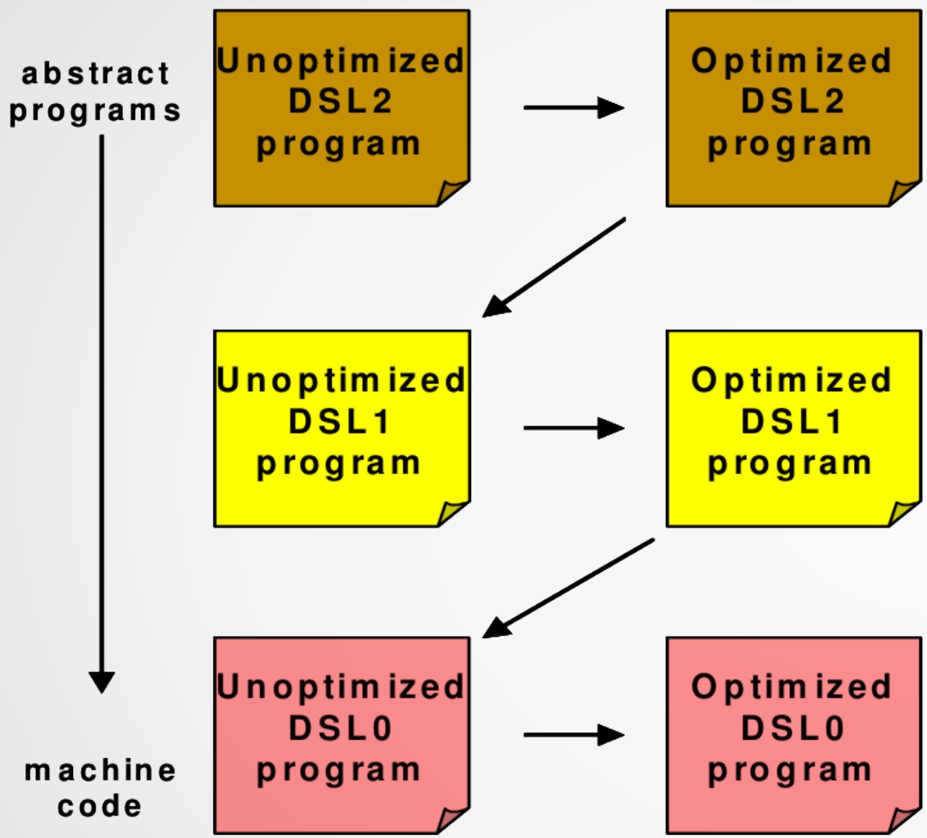



The Road to Utopia: A Future for Generative Programming, D. Batory, Domain-Specific Program Generation, LNCS 3016, Springer 2004



“Water does not flow uphill.” (Daoist saying)

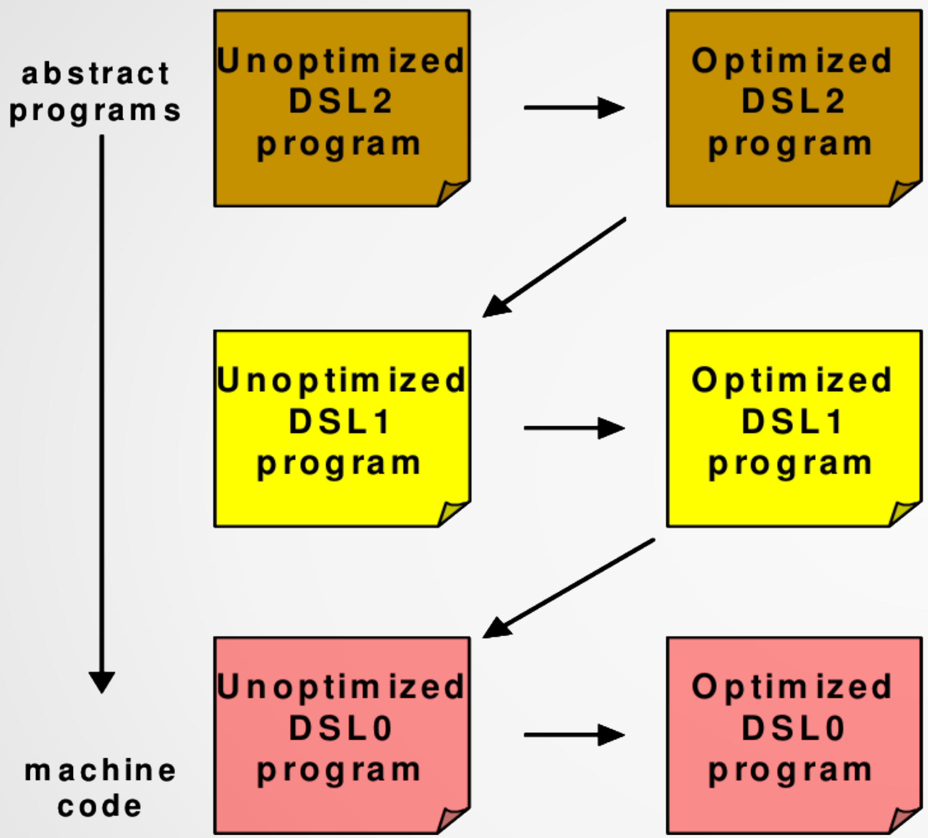
# Hierarchical DSL Optimization



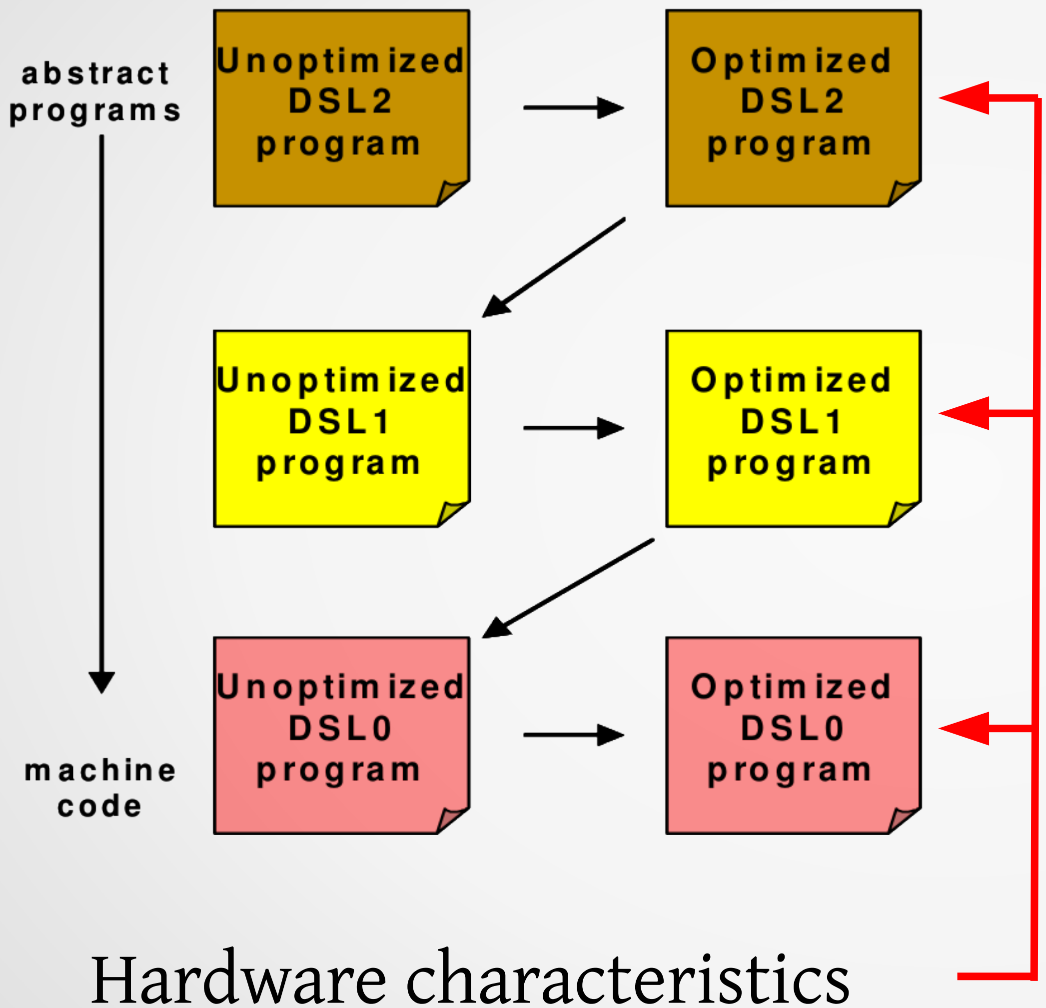

The Road to Utopia: A Future for Generative Programming, D. Batory, Domain-Specific Program Generation, LNCS 3016, Springer 2004


"Water does not flow uphill." (Daoist saying)

# Optimization w.r.t. the Hardware

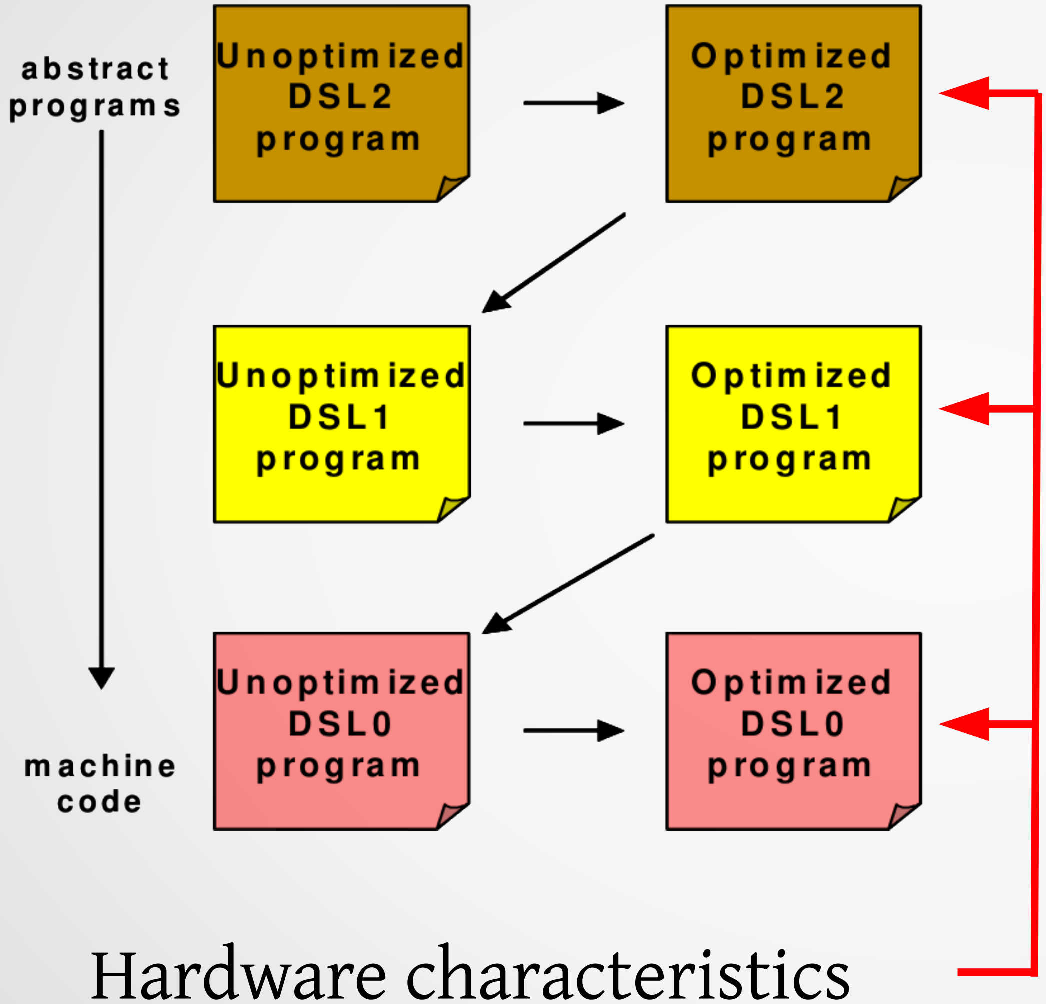


# Optimization w.r.t. the Hardware




Exploit hardware characteristics on every level!

# Optimization w.r.t. the Hardware




Exploit hardware characteristics on every level!

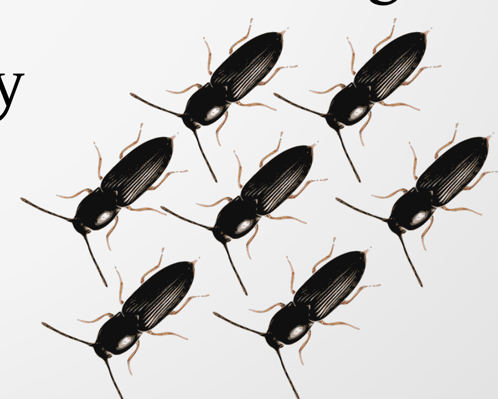
What cannot be optimized analytically becomes a parameter for auto-tuning or machine learning.

# Many, many DSLs?

- DSLs for stencils, dense linear algebra, sparse linear algebra, image processing, data parallel algorithms, work queues, parallel containers, ...
- Recently many papers with titles like „DSL (and run-time environment) for ...“ are published.

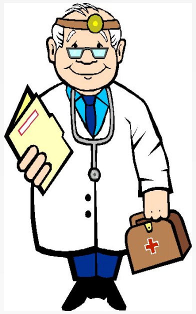
# Many, many DSLs?

- DSLs for stencils, dense linear algebra, sparse linear algebra, image processing, data parallel algorithms, work queues, parallel containers, ...
- Recently many papers with titles like „DSL (and run-time environment) for ...“ are published.
- But: compilers have bugs, Optimizers have even more bugs
- DSL compilers/optimizers likely to be buggy
- What does one do when things go wrong?



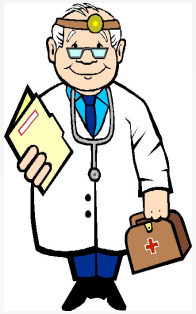
# Ask for a Second Opinion!

When you are not satisfied with the work of a particular expert...



# Ask for a Second Opinion!

When you are not satisfied with the work of a particular expert...



... you ask for a second opinion.

# Ask for a Second Opinion!

When you are not satisfied with the work of a particular expert...

... you ask for a second opinion.

You can do this with compilers, too.

# Ask for a Second Opinion!

When you are not satisfied with the work of a particular expert...

... you ask for a second opinion.

You can do this with compilers, too.

There may be as many "opinions" as "experts" (just try some implementations of OpenCL).

# Domain-specific vs. Standards

- Widely-used languages are standardized:

  C, C++, Java, OpenMP, MPI, OpenCL, ...

- Standardization takes time.
- We cannot expect several implementations of a particular DSL to be made.
- Polyhedral compilation:
  ~25 years to get a stable tool chain with release quality

# Challenges for Parallel DSL Engineering

- Tools for DSLs support parser, editor, (non-optimizing) compiler generation.
- Need support for optimizers
  - Optimization rules are usually complex
  - Abstractions (rewrite rules, etc.) help
- Can we find a “small” set of techniques that allow for the construction and verification of DSL optimizers?
- Can different DSLs and their optimizers be combined?

# The DAO of Parallel Software Construction

- Simplify your parallel programming: restrict to a **D**omain of the right size
- **A**nalysis: Find right parameters to tune (“small” search space)
- **O**ptimization of the parameters following the laws of the domain and the target hardware
- Challenges: tools for optimizers construction and composition

“The DAO is silent.” (Raymond Smullyan)