Shonan meeting

Reiji Suda

2014/05

My interest

- High performance computing
  - Parallel processing
- Numerical algorithms
  - Fast algorithms, parallel algorithms
- Autotuning
- Scheduling (for parallel processing)
- Scientific applications

We need code generator

- We are developing a code transformer based on ROSE compiler
  - Source to XML
  - Transform XML
  - XML to source
- Assuming existing (scientific) code, and adding directives for transforms
- Debug
  - Must be very difficult
  - Generated codes are visible

Want code generation (1/2)

- Effects of tuning techniques are:
  hard to predict, and
  dependent on data (size and value)
- Different choice for:
  different hardware, and
  different data set

- Combinations (and order) of optimizations
- Autotuning, parametric code generation

Want code generation (2/2)

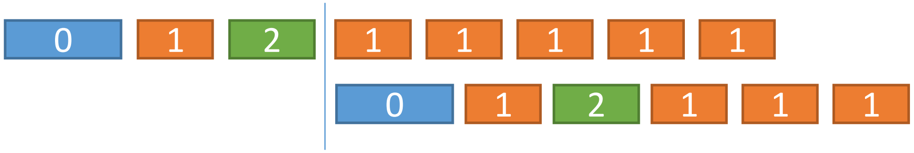
- Some compilers do very excellent optimizations
    - But sometimes does not expected optimizations…
- Data structure, and other global choice
- Sometimes optimization is not applied
    - Pointer aliasing and indirect array (e.g. A[B[i]])
    - Calling function compiled separately
    - Too long function, too short function
    - Conditional branches (too general function)
    - Dependency on data, variable (but actually constant)
    - Abstraction mechanisms (inheritance, method table)
    - Exceptions
    - Better choice unknown or data dependent
- Developer's knowledge
    - For example, B[i] are distinct

Autotuning

  - Now we cannot imagine which kind of code is best
  - So we are going this:
- Generate many candidate codes, variants (for one function)
- Run them, Measure performance
- Choose the best performed one

Offline and online autotuning

- Offline
  - Run the variants with sample data set
  - Choose the best performed one (static choice)



- Online
  - Choose one variant for each call of the function
  - Measure the performance and fix the next variant
- Parametric generation of variants
  - Runtime code generation in online autotuning

I tried metaocaml

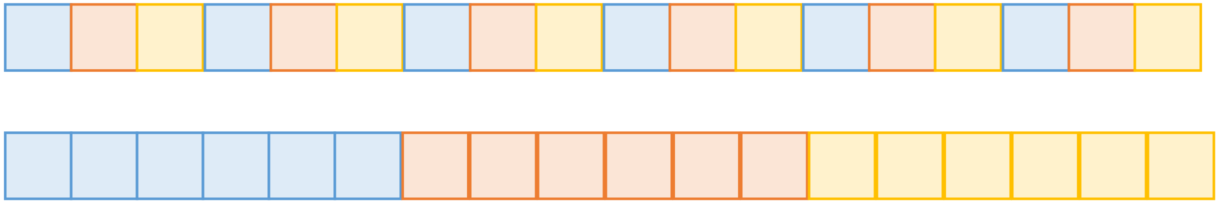
- let c x = .< 1 + 2 + x >.
  - Is there a way to make it .<3 + x>. ?
- let f = .<fun p x -> if p then sin x else cos x>.
  - Is there a way to fix p as true or false to get .<sin x>. or .<cos x>.?
  - Is there any way to fix x as 1.0 to get .<if p then sin 1.0 else cos 1.0>.?
  - Any beautiful way to revise the caller?
- How to apply basic transformation to existing code, or write generator based on existing code?
  - And to see the resulting code?

Array of structs / struct of arrays

```
typedef struct {
  double x, y, z;
} point;

point p[N];
```

```
struct {
  double x[N];
  double y[N];
  double z[N];
} p;
```

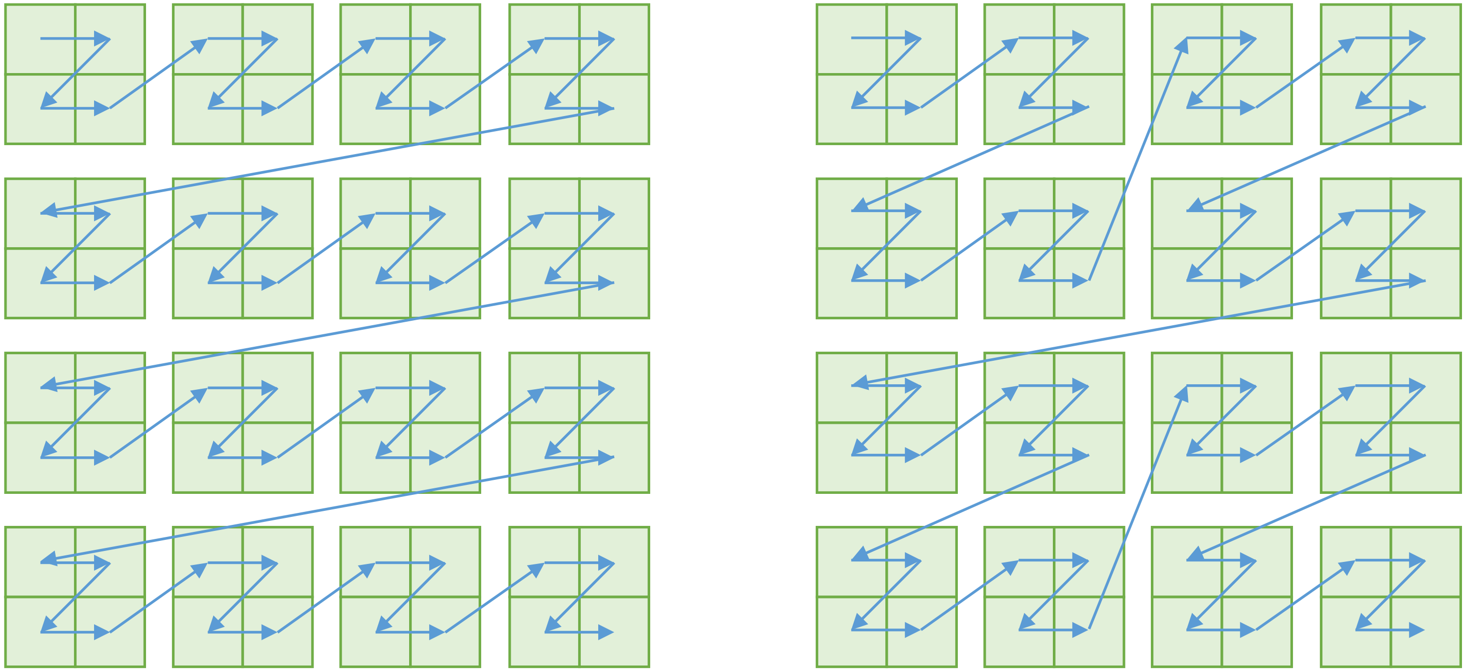


Array of structs

Struct of arrays

Case 1: increase x of all elements by 1
Case 2: compute norm sqrt(x*x + y*y + z*z) for each elements

Tiled data / Space-filling curve



Loops should be restructured accordingly (as much as possible)
For example: for all elements, row-wise, column-wise, diagonal, random…

Arrays

- Padding (cf. Prof. Takahashi’s presentation)
- Order of dimensions
  - Row-major A[M][N] or column-major A[N][M]
  - 3D: A[L][M][N], A[L][N][M], A[M][L][N], …
- Temporal copy (gather & scatter)
- Compression

- Scalar expansion / array temporal removal

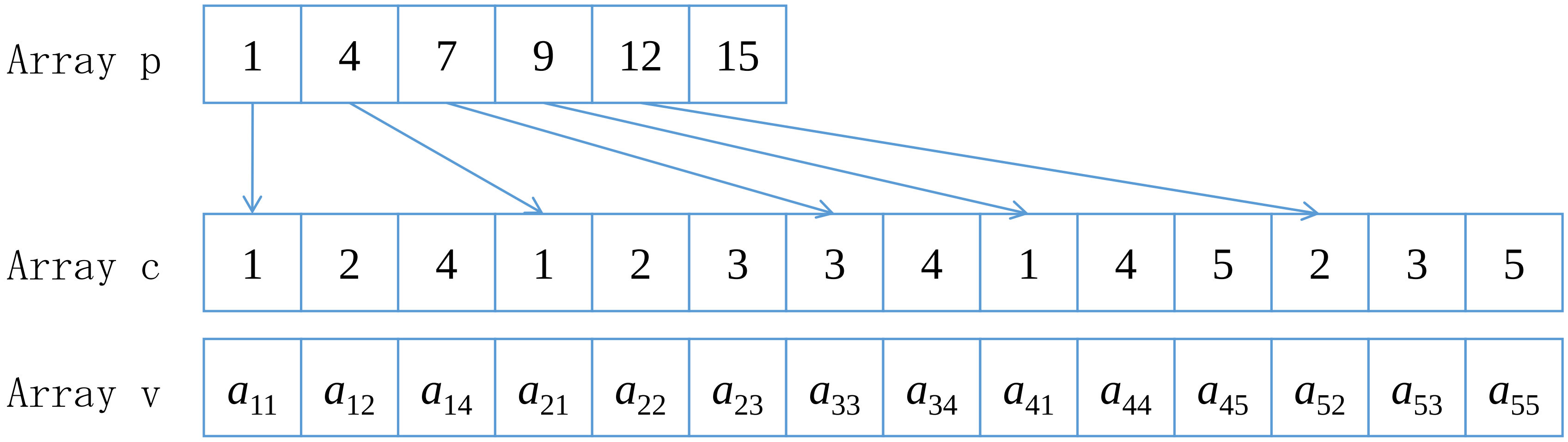
sparse matrix format（CRS）

Matrix A

a_{11}	a_{12}		a_{14}	
a_{21}	a_{22}	a_{23}		
		a_{33}		a_{34}
a_{41}			a_{44}	a_{45}
	a_{52}	a_{53}		a_{55}

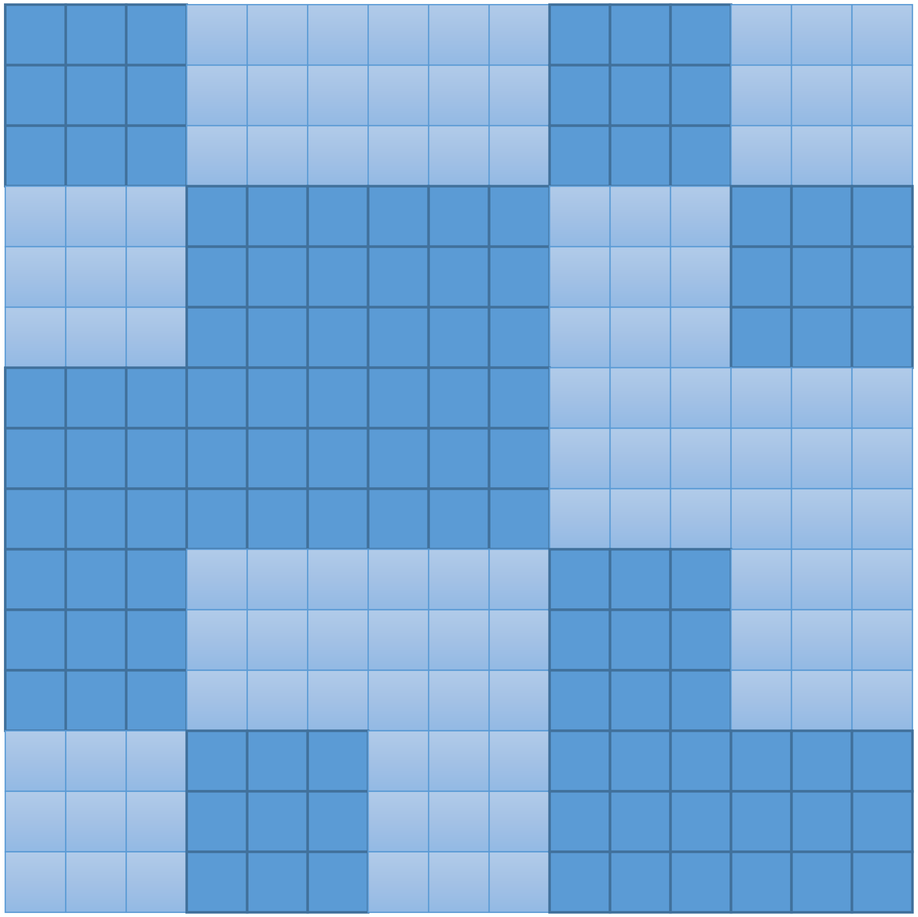
Empty = 0

Non-zero elements are stored

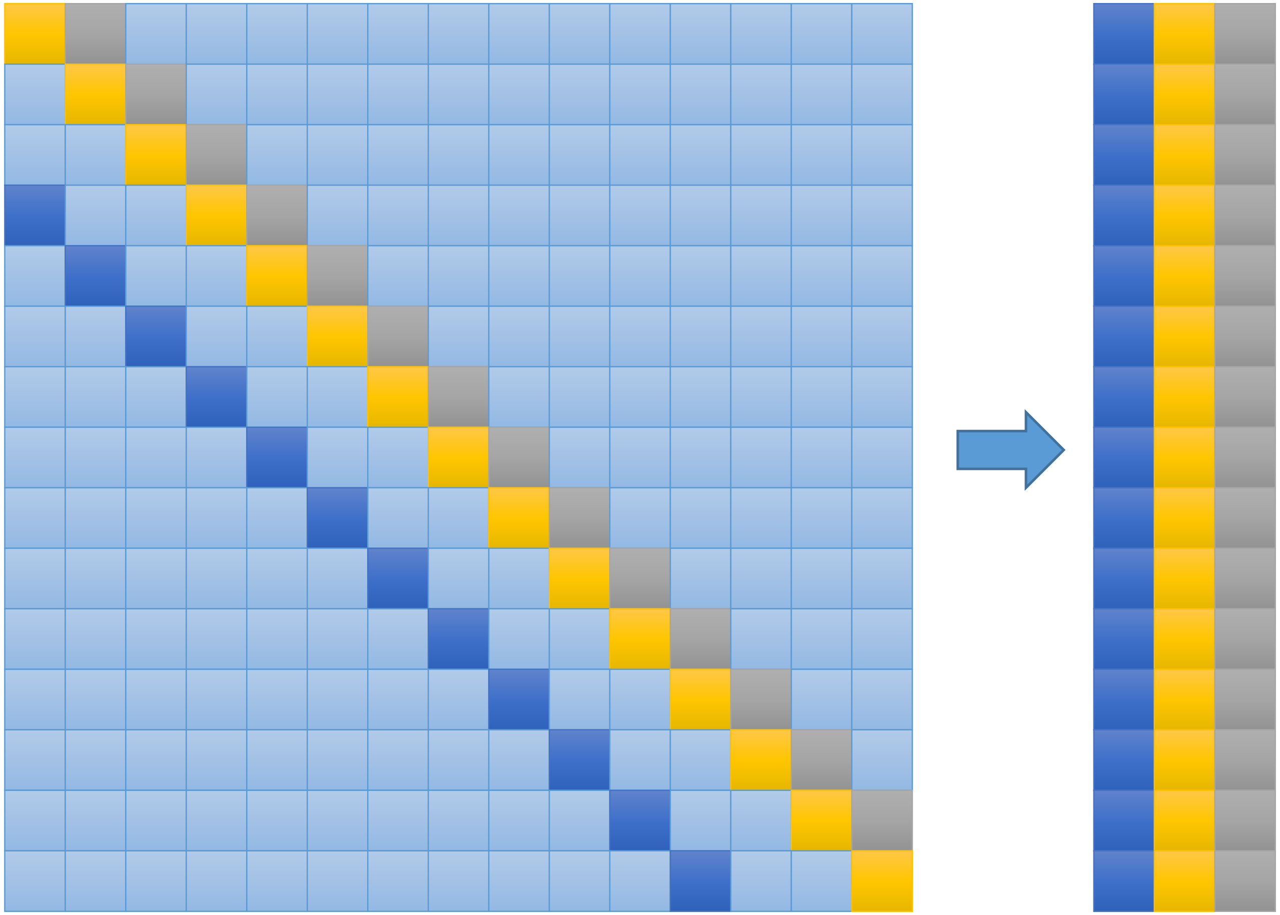


Array p	1	4	7	9	12	15								
Array c	1	2	4	1	2	3	3	4	1	4	5	2	3	5
Array v	a_{11}	a_{12}	a_{14}	a_{21}	a_{22}	a_{23}	a_{33}	a_{34}	a_{41}	a_{44}	a_{45}	a_{52}	a_{53}	a_{55}

BSC (3, 3)



Diagonal



Sparse matrix formats

- More data structures
- And any matrix A and be written

$$A = A_1 + A_2$$

  with A_1 and A_2 in different format
- Best format depends on the matrix and hardware

- Write one code and transform into for another format?
- (Semi)automatic format transformer?

Loop transformations

- Loop interchange
- Loop unrolling
- Loop tiling / stripmining (unroll-and-jam)
- Loop fusion / loop fission
- And more transformations
  - Software pipelining / pre-fetch and post-store
  - Loop peeling, index set splitting
  - Loop inversion

Function call

- Inline expansion / procedure extraction
- Loop inlining / loop embedding

<= Need enough operations to fill CPU pipeline

- Specialization, partial evaluation
  - Removing error check
  - Function may be too general
  - Full unrolling of loops
  - I/O buffering

Recomputation

- Recomputation
  <=> common subexpression elimination

- To reduce working set size
- To reduce total memory usage
  - E.g. need secondary storage
- To reduce dependency
  - Reduce communication
  - Enhance parallelism

Tuning for SIMD

- Vector length
    - Loop interchange, loop coalescing
    - Length multiple of hardware parallelism
- Divergence reduction
    - Minimize operations in “if”
    - Unit element (no effect)
- Parallelism
    - Atomic operation
    - Thread-private arrays
- Coalescing access
    - Structures of arrays, array dimensions
    - Padding, alignment

Algorithms

- Communication avoiding algorithms
  - Tall Skinny QR factorization
  - Communication-avoiding Krylov subspace method
- Stencil computations
  - Temporal blocking
  - and skewing or alternating
  - and communication latency hiding
- Collective communication algorithms

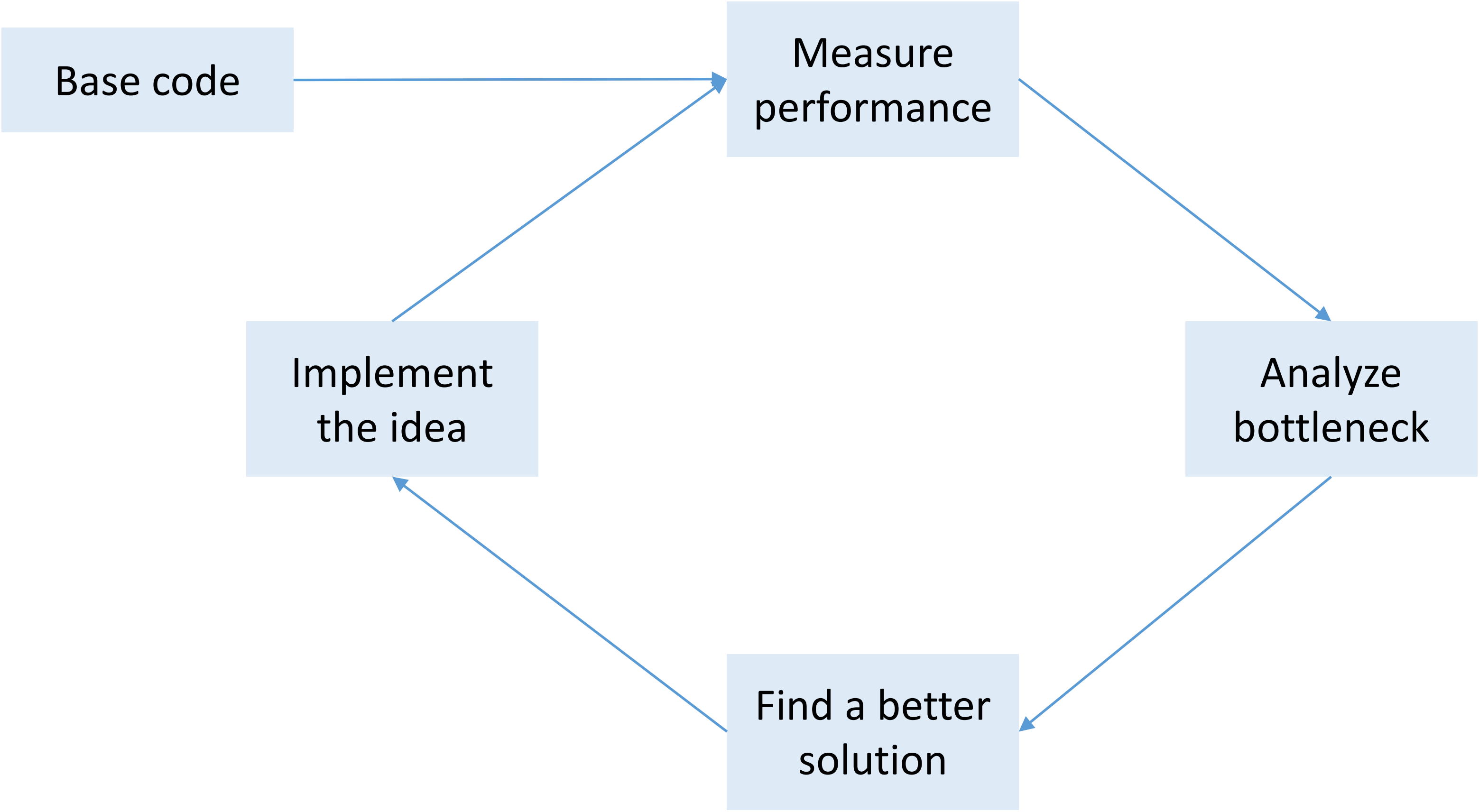
- How to generate high performance code with minimal modification to the original code?
    - Because hardware evolves so quickly, we will have to cancel the modification and introduce another
- Way of (quick and safe) removal of modification
- How to stage code and appropriate matching caller at once?

Code analysis?

- Is there any way to generate derivative (differential) of a given function?
  - Known as automatic differentiation

- Is there any way to check the output y is linear transformation of the input x?
  - That is, can be written as y = A x
- Is there any way to check the above matrix A is symmetric?

Optimization by compiler or user?



Reasons of “better solutions”

- Compilers cannot optimize code perfectly
  - Cf. full employment theorem for compiler writers
- Optimizations unsafe in general
- Optimizations effective for specific datasets
- Optimizations not implemented in the compiler yet

- HPC people will want to control their optimizations for ever