Redesigning Codes to Extract Parallelism

Jinzhen Wang
jwang96@charlotte.edu

Department of Computer Science
UNC Charlotte

Learning Outcomes

Lecture

At the end of this session you will know how to

- Rewrite sequential codes to expose more parallelism

What to do when the code is not parallel enough?

Sometimes more art than science!

There are techniques to do some things.
Eventually it devolves in algorithm design.

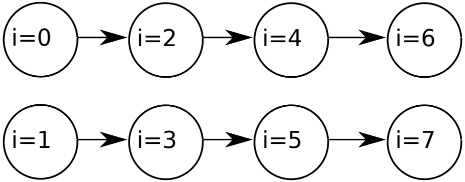
Answering questions like

- Does it matter if these two things are swapped?
- Would the code still be correct if... ?
- Can we do something completely different?
- Is there a different expression that can compute the same value?

Loop Splitting

Example

```
void red () {
  for (int i=0; i<N; ++i) {
    val[i] = f(i);
    re[i%2] += val[i];
  }
}
```

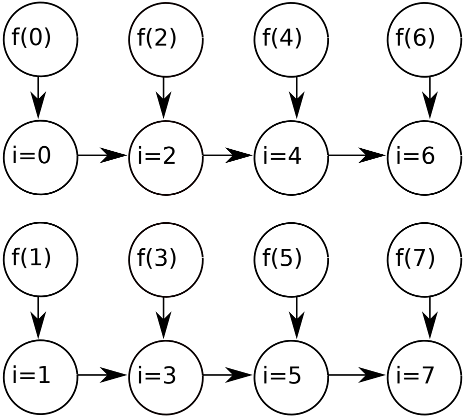


Loop Splitting

Do it differently

```
void red () {
  for (int i=0; i<N; ++i)
    val[i] = f(i);
  for (int i=0; i<N; ++i)
    re[i%2] += val[i];
}
```

- Changes dependency structures.
- Useful if `f(i)` is expensive.
- Assumes `f(i)`s are independent.



Scoping

Matvec

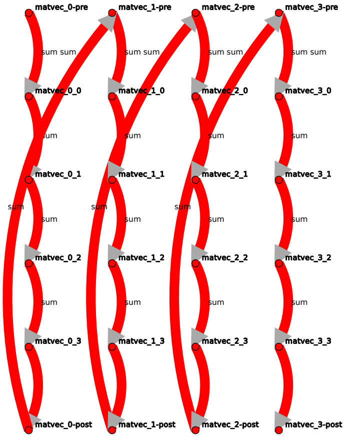
```
float A[N][N];
float x[N];
float y[N];

void matvec() {
  float sum;

  for (int i=0; i<N; ++i) {
    sum = 0.;

    for (int j=0; j<N; ++j)
      sum += A[i][j] * x[j];

    y[i] = sum;
  }
}
```



No parallelism because of sum

Scoping

Matvec

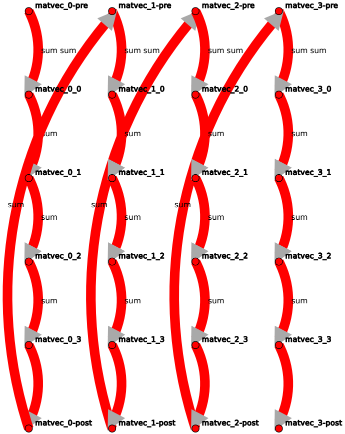
```
float A[N][N];
float x[N];
float y[N];

void matvec() {
  float sum;

  for (int i=0; i<N; ++i) {
    sum = 0.;

    for (int j=0; j<N; ++j)
      sum += A[i][j] * x[j];

    y[i] = sum;
  }
}
```



No parallelism because of sum

Work: $N * (N + 2) = \Theta(N^2)$

Width: $1 = O(1)$

CP: $N * (N + 2) = \Theta(N^2)$

Scoping

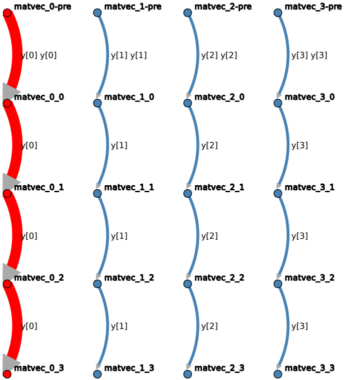
Matvec

```
float A[N][N];
float x[N];
float y[N];

void matvec() {

  for (int i=0; i<N; ++i) {
    y[i] = 0.;

    for (int j=0; j<N; ++j)
      y[i] += A[i][j] * x[j];
  }
}
```



Removing the dependency on sum (or moving sum into the scope of the i loop.) reveals parallelism.

Scoping

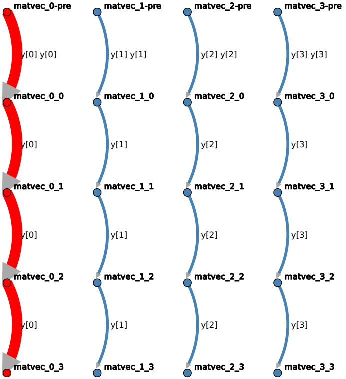
Matvec

```
float A[N][N];
float x[N];
float y[N];

void matvec() {

  for (int i=0; i<N; ++i) {
    y[i] = 0.;

    for (int j=0; j<N; ++j)
      y[i] += A[i][j] * x[j];
  }
}
```

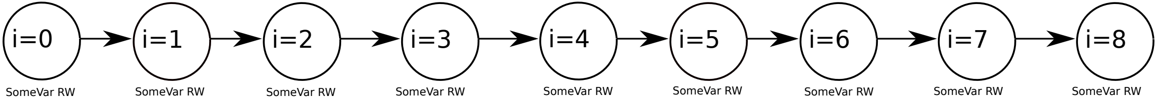


Removing the dependency on `sum` (or moving `sum` into the scope of the `i` loop.) reveals parallelism.
Work: $N * (N + 1) = \Theta(N^2)$
Width: $N = \Theta(N)$
CP: $N + 1 = \Theta(N)$

Targetting a particular number of processors

Problem: One variable causes a chain of dependency

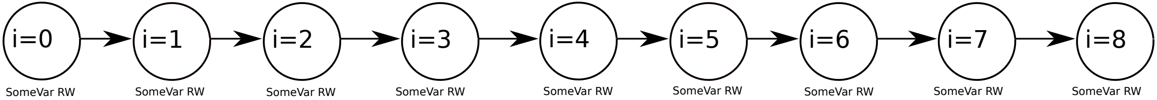
There are cases where a code's PTG is a chain because of a single variable.



Targetting a particular number of processors

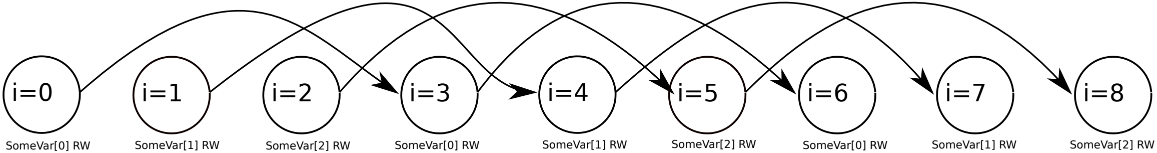
Problem: One variable causes a chain of dependency

There are cases where a code's PTG is a chain because of a single variable.



An idea: Introducing *P* new variables

Maybe we can introduce *P* copies of that variables in the hope of creating *P* chains of tasks.

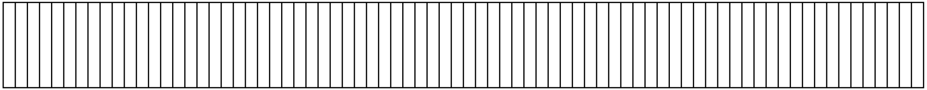


Probably that code would be incorrect, but maybe it is fixable by adding a final task at the end.

Targetting a particular number of processors

Problem: a bunch of work all interdependent

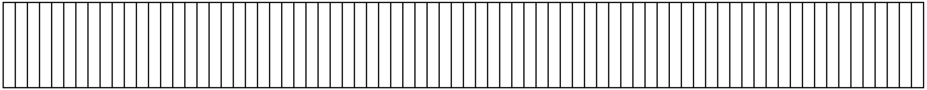
There are cases where a code's PTG is a long chain with complex variables accesses



Targetting a particular number of processors

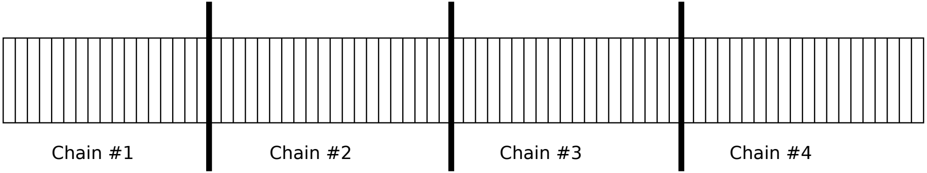
Problem: a bunch of work all interdependent

There are cases where a code's PTG is a long chain with complex variables accesses



Idea: Naively split the work in P chains

A common strategy is to naively split the work in P chunks arbitrarily.



And ask "Why doesn't that work?" May need some adjustement, maybe some particular cuts are easier. Maybe the algorithm can be adjusted.

Or...

Or sometimes you need to do things completely differently
Involves rewriting the algorithm from scratch.
No general rule on that: Look at the dependencies and see how you can do something else.

External

Dependency extraction:

- The origins of the model: A. J. Bernstein. Analysis of programs for parallel processing. IEEE Transactions on Electronic Computers, 15:757–762, Oct. 1966.
- Voevodin V., Antonov A., Voevodin V. (2018) What Do We Need to Know About Parallel Algorithms and Their Efficient Implementation?. In: Prasad S., Gupta A., Rosenberg A., Sussman A., Weems C. (eds) Topics in Parallel and Distributed Computing.
- Chapter 2 to 5.1 of Oliver Sinnen. Task Scheduling for Parallel Systems. John Wiley & Sons, Inc. 2007. Access it through the library: `https://librarylink.uncc.edu/login?url=https://onlinelibrary.wiley.com/doi/book/10.1002/0470121173`
- Chapter 1 and 7 of. H. Casanova, A. Legrand, Y. Robert. Parallel Algorithms, CRC Press. 2008

Software:

- Par graph lib: `https://github.com/esaule/par_graph_lib`
- Cilk Plus extract dependencies with the programmers help: `https://software.intel.com/en-us/node/522598`
- Athapascan/KAAPI does something similar: `https://hal.inria.fr/inria-00069901/document`

Typical compiler optimization:

- Loop fission: `https://en.wikipedia.org/wiki/Loop_fission`
- Loop tiling: `https://en.wikipedia.org/wiki/Loop_tiling`
- Various: `https://en.wikipedia.org/wiki/Compiler_optimization`