

crossref.bib

Scheduling and Amortization for MPC]Scheduling and Amortization for MPC

levyb3@rpi.edu Rensselaer Polytechnic Institute Troy New York

shernb@rpi.edu Rensselaer Polytechnic Institute Troy New York

kennal@rpi.edu Rensselaer Polytechnic Institute Troy New York

milanova@cs.rpi.edu Rensselaer Polytechnic Institute Troy New York

This work was done in part while the author was at RPI. m.ishaq@ed.ac.uk University of Edinburgh Edinburgh Scotland

This work was done in part while the author was visiting UCLA and supported in part by DARPA and SPAWAR

10003752.10010124.10010138.10010143 < /concept_id >< concept_desc > Theory of computation

[500]Theory of computation Program analysis [500]Theory of computation Cryptographic protocols [300]Security and privacy

protocol mixing; linear programming; multiparty computation; program analysis; cryptography

Correctness Argument

Towards Extension of Basic Vectorization

Removal of Infeasible Edges

Array writes limit vectorization as they sometimes introduce infeasible loop-carried dependencies. Consider the following code:

```
for i in range(N): A[i] = B[i] + 10; B[i] = A[i] * D[i-1]; C[i] = A[i] * D[i-1]; D[i] = B[i] * C[i];
```

In Cytron's SSA this code (roughly) translates into

for i in range(N): 1. $A_0 = \phi(A, A_1)$ 2. $B_0 = \phi(B, B_1)$ 3. $C_0 = \phi(C, C_1)$ 4. $D_0 = \phi(D, D_1)$ 5. $A_1 = A_0$; $A_1[i] = B_0[i] + 10$; $C_1 = C_0$; $C_1[i] = A_1[i] * D_0[i-1]$; $D_1[i] = B_0[i] * C_1[i]$

There is a cycle around $B_0 = \phi(B, B_1)$ that includes statement $A_1[i] = B_0[i] + 10$; and that statement won't be vectorized.

The following algorithm removes certain infeasible loop-carried dependencies that are due to array writes. Consider each array A written in loop j including enclosed loops in j. $dep = \text{False}$ each pair (i, k) def: $A_m[f(i, j, k)] = \dots$, and use:

Ishaq: Note to self: This algorithm is an instantiation for j loop, the one for k loop will be exactly the same, modulo j.

Consider a loop j enclosed in some fixed i. Only if an update (definition) $A_m[f(i, j, k)] = \dots$ at some iteration j ref: $A_m[f(i, j, k)] = \dots$

Consider the earlier example. There is a single loop, i. Clearly, there is no pair i and i', where $i < i'$ that make $i = i'$.

Ana: Commented out previous writeup. Removal/handling of targetless phi-nodes will go into the Extension to Basic Vectorization.

Array MUX refinement

Ana: TODO: I think we should implement this.

Next, the algorithm refines array MUX statements. MPC-source after Cytron's SSA may result in statements $A_j = \text{MUX}(\dots, A_k, A_l)$

This is to reduce the dimensionality of simd-ified computation. Technically, $A_j = \text{MUX}(\dots, A_k, A_l)$ is a simdified operation.

each $stmt: A_j = \text{MUX}(c, A_k, A_l)$ in the MPC-source seq. $i_1 = \text{find_update}(A_k)$ Is null when $A_k = \phi(\dots)$ $i_2 = \text{find_update}(A_l)$

$A_j = A_{j-1}$; $A_j[i_1] = \text{MUX}(c, A_k[i_1], A_l[i_1])$ $stmt$ stays as is

Extension of Basic Vectorization with Array Writes

Ana: TODO: Handling of Targetless phi nodes should be done here, as an extension to Basic Vectorization.

Restricting Array Writes

For now, we restrict array updates to *canonical updates*. Assume (for simplicity) a two-dimensional array $A[I, J]$. A canonical update is

The update $A[i, j]$ can be nested into an inner loop and there may be multiple updates, i.e., writes to $A[i, j]$. However, reads through an arbitrary formula, such as $A[i-1]$ for example, are allowed, however, we assume the programmer

Changes to Basic Vectorization

One change to Basic vectorization is the expansion of dimension if the array write or read occurs in a nested loop. That is, if

The other change concerns def-use edges $X \rightarrow Y$ where X defines and Y uses an array variable (e.g., the definition of X is at the same level as Y). We do nothing, just propagate the array, which happens to be of the right dimension. **Ana: There is a change to inner-to-outer $X \rightarrow Y$. Let A be the array defined at X . If the dimensionality of A is greater than its canonical dimensionality, then $X \rightarrow Y$ is outer-to-inner. Add $\text{raise_dim}(\dots)$ (at X) as in Basic Vectorization.**

"mixed" $X \rightarrow Y$. We assume that the mixed edge is transformed into an inner-to-outer followed by outer-to-inner edge.

Examples with Array Writes

Example 1

Recall that after removal of infeasible edges and redundant phi-nodes, the Aiken's array write example will be (roughly) as follows:

```
for i in range(N): 1.  $D_0 = \phi(D, D_1)$  2.  $A_1 = A$ ;  $A_1[i] = B[i] + 10$ ; 3.  $B_1 = B$ ;  $B_1[i] = A_1[i] * D_0[i-1]$ ; 4.  $C_1 = C$ ;  $C_1[i] = A_1[i] * B_1[i]$ ; 5.  $D_1[i] = B_1[i] * C_1[i]$ 
```

There are no nested loops, thus, array accesses remain as is.

After Phase 1 of Basic vectorization we have the following code: for i in range(N): 1. $D_0 = \phi(D, D_1)$ 2. $A_1 = A$; $A_1[i] = B[i] + 10$; 3. $B_1 = B$; $B_1[i] = A_1[i] * D_0[i-1]$; 4. $C_1 = C$; $C_1[i] = A_1[i] * B_1[i]$; 5. $D_1[i] = B_1[i] * C_1[i]$

In Phase 2 we get rid of the arrows on statements 3, 4, and 5 as they are not vectorizable and in Phase 3 we add SIMD to the code:

1. $A_1 = A$; $A_1[\vec{i}] = \text{ADD_SIMD}(B[\vec{i}], [10, \dots])$ Fully vectorized, size N. FOR i=0; i<N; i++; MOTION loop 2. $D_0 = \phi(D, D_1)$

Consider MPC-source of Histogram after removal of infeasible edges and redundant phi-nodes, and MUX refinement:

res!1 = [] for i: plaintext in range(0, num_bins): res!2 = $\phi(\text{res!1}, \text{res!3})$... res!4 = res!2 Added due to the "mixed" edge

After Phase 1 of Basic Vectorization:

res!1 = [] for i: plaintext in range(0, num_bins): res!2 = $\phi(\text{res!1}, \text{res!3})$... res!4 = res!2 Added due to the "mixed" edge

After Phase 2 and Phase 3. The EQ operation is vectorizable across both dimensions, and the rest of the computation is as follows:

... !2!3[\vec{i}, \vec{j}] = $\text{EQ_SIMD}(A[\vec{i}, \vec{j}], [[0, 0, \dots], [1, 1, \dots], \dots, [\text{num_bins}-1, \text{num_bins}-1, \dots]])$ FOR j=0; j<N; j++; MOTION loop

Divide-and-Conquer

Ana: TODO: Now that we have broken FOR loops into smaller chunks, we can add Divide-and-conquer reasoning to the code.

Implementation and Evaluation

Future Work

Conclusions