$_{c}rossref.bib$

Scheduling and Amortization for MPC]Scheduling and Amortization for MPC

levyb3@rpi.edu Rensselaer Polytechnic Institute Troy New York

shermb@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 Sco

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

¡ccs2012¿ ¡concept¿ ¡concept$_{i}d$ > 10003752.10010124.10010138.10010143 < $/concept_{i}d >< concept_{d}esc > Theoryof$

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

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 followin
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. $\mathtt{A}_0 = \phi(\mathtt{A}, \mathtt{A}_1)$ 2. $\mathtt{B}_0 = \phi(\mathtt{B}, \mathtt{B}_1)$ 3. $\mathtt{C}_0 = \phi(\mathtt{C}, \mathtt{C}_1)$ 4. $\mathtt{D}_0 = \phi(\mathtt{D}, \mathtt{D}_1)$ 5. $\mathtt{A}_1 = \mathtt{A}_0$; $\mathtt{A}_1$[i] = $\mathtt{B}_0$[i] + 10; e
There is a cycle around $\mathtt{B}_0 = \phi(\mathtt{B}, \mathtt{B}_1)$ that includes statement $\mathtt{A}_1$[i] = $\mathtt{B}_0$[i] + 10; and that statement won't be vecto
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 = False each pair def: $\mathtt{A}_m[f(i,j,k)]$ = ..., and use:
Ishaq: Note to self: This algorithm is an instantiation for j loop, the one for k loop will be exactly the same, modu
Consider a loop j enclosed in some fixed $\underline{i}$. Only if an update (definition) $\mathtt{A}_m[f(i,j,k)]$ = ... at some iteration $\underline{j}$ refe
Consider the earlier example. There is a single loop, i. Clearly, there is no pair $\underline{i}$ and $\underline{i}'$, where $\underline{i} < \underline{i}'$ that make $\underline{i}$ =
Ana: Commented out previous writeup. Removal/handling of targetless phi-nodes will go into the Extension to Bas

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 $\mathtt{A}_j$ =
This is to reduce the dimensionality of simd-ified computation. Technically, $\mathtt{A}_j = MUX(..., \mathtt{A}_k, \mathtt{A}_l)$ is a simdified oper
each *stmt*: $\mathtt{A}_j = MUX(c, \mathtt{A}_k, \mathtt{A}_l)$ in the MPC-source seq. $i_1 = find\_update(A_k)$ Is null when $A_k = \phi(...)$ $i_2 = find\_up$
$\mathtt{A}_j = \mathtt{A}_{j-1}$; $\mathtt{A}_j[i_1] = MUX(c, \mathtt{A}_k[i_1], \mathtt{A}_l[i_1])$ *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 $\mathtt{A}[I, J]$. A can
The update $\mathtt{A}[i, j]$ can be nested into an inner loop and there may be multiple updates, i.e., writes to $\mathtt{A}[i, j]$. Howev
Reads through an arbitrary formula, such as $\mathtt{A}[i - 1]$ 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
The other change concerns def-use edges $X \to Y$ where X defines and Y uses an array variable (e.g., the definition
same-level $X \to Y$. We do nothing, just propagate the array, which happens to be of the right dimension. Ana: There n
inner-to-outer $X \to Y$. Let A be the array defined at X. If the dimensionality of A is greater than its canonical dimens
outer-to-inner $X \to Y$. Add *raise_dim(...)* (at X) as in Basic Vectorization.
"mixed" $X \to Y$. 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 (roug
for i in range(N): 1. $\mathtt{D}_0 = \phi(\mathtt{D}, \mathtt{D}_1)$ 2. $\mathtt{A}_1 = \mathtt{A}$; $\mathtt{A}_1$[i] = B[i] + 10; 3. $\mathtt{B}_1 = \mathtt{B}$; $\mathtt{B}_1$[i] = $\mathtt{A}_1$[i] * $\mathtt{D}_0$[i-1]; 4. $\mathtt{C}_1 = \mathtt{C}$; $\mathtt{C}_1$[i] = A
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. $\mathtt{D}_0 = \phi(\mathtt{D}, \mathtt{D}_1)$ 2. $\mathtt{A}_1 = \mathtt{A}$; $\mathtt{A}_1[\vec{i}]$
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 SI
1. $\mathtt{A}_1 = \mathtt{A}$; $\mathtt{A}_1[\vec{i}] = ADD\_SIMD(\mathtt{B}[\vec{i}], [10, ...])$ Fully vectorized, size N. FOR i=0; i¡N; i++; MOTION loop 2. $\mathtt{D}_0$ =
Example 2
Consider MPC-source of Histogram after removal of infeasible edges and redundant phi-nodes, and MUX refinemen
res!1 = [] for i: plaintext in range(0, num_bins): res!2 = ϕ(res!1, res!3) ... res!4 = res!2 Added due to the "mix
After Phase 1 of Basic Vectorization:
res!1 = [] for i: plaintext in range(0, num_bins): res!2 = ϕ(res!1, res!3) ... res!4 = res!2 Added due to the "mix
After Phase 2 and Phase 3. The EQ operation is vectorizable across both dimensions, and the rest of the computat
... !2!3$[\vec{i}, \vec{j}] = EQ_SIMD(\mathrm{A}[\vec{i}, \vec{j}]$, [[0,0...], [1,1...],...[num_bins-1,num_bins-1...]]) FOR j=0; j¡N; j++; MOTION loo
Divide-and-Conquer
Ana: TODO: Now that we have broken FOR loops into smaller chunks, we can add Divide-and-conquer reasoning
Implementation and Evaluation
Future Work
Conclusions