Automated Derivation of Parametric Data Movement Lower Bounds for Affine Programs

Olivry et al
The approach

- They present some algorithms to compute I/O complexity lower bounds (same setup as HBL)
  - Their lower bounds are justified using arguments about standard computation graphs, which they call CDAGs
  - They generate them from a polyhedral model representation, though

Two types of lower bounds:

- HBL-style
- A way of counting a lot of live variables

Also there's something about "may-spill sets"
CDAGs

Parameters: N, M;
Input: A[N], C[M]; Output: A[N];
for (t=0; t<M; t++)
  for (i=0; i<N; i++)
    A[i] = A[i] * C[t];

(a) C-like code

Parameters: N, M;
Input: A[N], C[M]; Output: S_{M-1}[N];
for (0 ≤ t < M and 0 ≤ i < N)
  if (t==0): S_{0,i} = A[i] * C[0];
  else: S_{t,i} = S_{t-1,i} * C[t];

(b) Corresponding single assignment form

(c) Corresponding CDAG. Input nodes A[N] (resp. C[N]) are shown in grey (resp. white) while compute node are shown in black

Fig. 1. Example 1
Pebble Games and S+T partitioning

Model computation with communication by playing a "pebble game" on the CDAG (place a red pebble to mean "in fast memory," blue to mean "in slow memory"; only have a fixed number of red pebbles to use)

Translate this into a sequence of computes and loads

If you have a fast memory of size S, and you prove that any k iterations require S+T inputs, you can partition the whole computation into N/k pieces and each must contain at least T loads
for \((0 \leq t < M \text{ and } 0 \leq i < N)\)

\[\text{if } (t==0) : \quad S[0,i]=A[i] \times C[0];\]
\[\text{else} : \quad S[t,i]=S[t-1,i] \times C[t];\]

\[D_A = [N] \rightarrow \{A[i] : 0 \leq i < N\}\]
\[D_C = [N] \rightarrow \{C[t] : 0 \leq t < M\}\]
\[D_S = [M \times N] \rightarrow \{S[t, i] : 0 \leq t < M \land 0 \leq i < N\}\]
\[|D_S| = MN\]

(a) Single assignment form

(c) Node domains

\[R_{e_1} = [N] \rightarrow \{A[i] \rightarrow S[0, i] : 1 \leq i < N\}\]
\[R_{e_2} = [M \times N] \rightarrow \{C[t] \rightarrow S[t, i] : 0 \leq t < M \land 0 \leq i < N\}\]
\[R_{e_3} = [M \times N] \rightarrow \{S[t, i] \rightarrow S[t+1, i] : 0 \leq t < M-1 \land 0 \leq i < N\}\]

(b) DFG

(d) Edge relations

Fig. 2. DFG for Example 1

Image taken directly from Olivry et al.
Using the DFG

- Edges are direct dependencies, and are affine
- Paths/walks are composed dependencies

Find some paths/walks and argue that they correspond to projections between sets of iterations and sets of memory accesses; then apply HBL

- They identify two types of paths for which they can argue this

\[(2) \text{ Use the DFG representation to find a subset } V' \subseteq V \text{ and a set of projections (group homomorphisms) } \phi_1, \ldots, \phi_m \text{ with the property that:}\]

\[\text{Any } K\text{-bounded set } P \subseteq V' \setminus \text{Sources}(V') \text{ satisfies } |\phi_j(\rho(P))| \leq K. \quad (4)\]

\[(3) \text{ Using Theorem 3.10, derive an upper bound } U \text{ on } |\rho(P)| \text{ for any } K\text{-bounded } P. \text{ This provides a lower bound } \left\lceil \frac{|V'\setminus\text{Sources}(V')|}{U} \right\rceil \text{ on the number } h \text{ of disjoint } K\text{-bounded sets in } V' \setminus \text{Sources}(V').\]

(More details here)
Using the DFG part 2

They also have a totally unrelated second lower bound they compute.

If there are two disjoint sets of vertices in the CDAG $V_1$ and $V_2$, where everything in $V_1$ has a path to $V_2$, and I can find $k$ disjoint paths between $V_1$ and $V_2$, then I need to do at least $k-S$ loads from slow memory (if my fast memory has size $S$).

They find some special cases of this using the DFG.

This is a bit strange and could maybe be extended usefully?
"May-spill sets"

Disclaimer: I don't fully understand this

General idea: If you have a fast memory of size S, you want to compute N iterations, and you know (through HBL, say) that only k iterations are computable given 2S values, you know you must have at least \((N/k)S\) memory accesses total.

In effect, this argument "partitions" the CDAG into \((N/k)\) disjoint sets of iterations, gets a lower bound (HBL-style, maybe) for each set, and sums them together.

Can I also sum lower bounds together if the sets are not disjoint?

Olivry et al say: yes, if their "may-spill sets" are disjoint.
"May-spill sets"

For a set \( V \) of iterations, its "may-spill set" is the set of iterations which have

1. both at least one incoming and at least one outgoing edge contained in \( V \), or

2. at least 2 outgoing edges contained in \( V \)

Intuitively: The set of nodes in \( V \) which "may spill" their value to slow memory if we only have to compute \( V \)

No communication with slow memory at all in "no-spill" set; no danger of double-counting
Questions for ourselves

- Can we generalize this to get computable lower bounds on communication in parallel?
- Can we use a more general model (and thus get stronger bounds on a wider range of algorithms)?
- Since we know that expanding and low-diameter graphs have large lower bounds in parallel hardware, can we get characterizations of which polyhedral algorithms have these properties?