15754 Project

Salil Joshi & Cyrus Omar

April 27, 2011

1 Grammar

We will work on a language that is essentially a simple subset of CUDA.
Types:

$$\begin{aligned} \text{T'} &:= \text{uint} \mid \text{int} \mid \text{float} \mid \text{bool} \\ \text{T} &:= \textbf{global}\ T'[] \mid \textbf{local}\ T'[] \mid T' \mid \textbf{const}\ T' \mid \textbf{TID}(a, b) \end{aligned}$$

Terms

$$e := x \mid 1 \ldots \mid tid \mid bid \mid a[e]$$

$$\text{stmt} := \langle\rangle \mid \text{cmd}; \text{stmt}$$

$$\begin{aligned} \text{cmd} := & \textbf{if}\ e\ \textbf{then}\ \text{stmt}\ \textbf{else}\ \text{stmt} \\ & \mid \textbf{for}\ x\ \textbf{in}\ [e, e, e]\{\text{stmt}\} \\ & \mid \textbf{sync} \\ & \mid x := e \\ & \mid a[e] := e \\ & \mid \textbf{decl}\ T\ x\ \textbf{in}\ \text{stmt} \end{aligned}$$

The language is a simple imperative language with a few special features:

- *tid* is special variable that holds the thread index of the current thread. Similarly, *bid* holds the block index.
- Arrays are either **global** or **local**. **global** arrays are stored on the global device memory while **local** arrays are stored on the block level shared memory
- A block level synchronization primitive, **sync** which corresponds to CUDA's `__syncthreads__`
- **const** types. The value of an expression of this type can be computed at compile time.
- **TID**(a, b). An expression of this type has the value $a * \text{tid} + b$ i.e. it is an affine transformation of the thread index

2 Typing

We have three typing judgements:

- $\Gamma/\Delta \vdash e : T$ for expressions e states that e has type T in the context Γ. The context Δ collects the set of indices used by e for every global array that e accesses (reads from). However, only those indices which are *affine transformations of the thread index* (i.e. have type $\mathbf{TID}(a, b)$) are collected. Δ will later be used to determine the memory segments that are accessed by the program, which is useful for optimizations like memory coalescing and data sharing. At present we focus on memory coalescing.
- $\Gamma/\Delta_r/\Delta_w \vdash C$ ok for commands C states that the command is well formed and all subexpressions are well typed. In addition, Δ_r is the set of global array indices *read from* by C and Δ_w is the set of global array indices *written to* by C, similar to Δ in the previous judgement. Once again we only track array indices that are affine transformation of the thread index.
- $\Gamma/\Delta_r/\Delta_w \vdash S$ ok similar to the previous judgement, but for statements.

2.1 Expressions

$$\frac{x : T \in \Gamma}{\Gamma/\cdot \vdash x : T} \qquad \frac{}{\Gamma \vdash \text{tid} : \mathbf{TID}(1, 0)} \qquad \frac{}{\Gamma \vdash \text{bid} : \text{uint}}$$

$$\frac{\Gamma/\Delta_1 \vdash e_1 : \mathbf{TID}(a_1, b_1) \qquad \Gamma/\Delta_2 \vdash e_2 : \mathbf{TID}(a_2, b_2)}{\Gamma/\Delta_1 \cup \Delta_2 \vdash e_1 + e_2 : \mathbf{TID}(a_1 + a_2, b_1 + b_2)} \qquad \frac{\Gamma/\Delta_1 \vdash e_1 : \mathbf{TID}(0, c,) \qquad \Gamma/\Delta_2 \vdash e_2 : \mathbf{TID}(a, b)}{\Gamma/\Delta_1 \cup \Delta_2 \vdash e_1 * e_2 : \mathbf{TID}(a * c, b * c)}$$

$$\frac{\Gamma \vdash a : \mathbf{global}\ T'[] \qquad \Gamma/\Delta \vdash e : \text{uint}}{\Gamma/\Delta \vdash a[e] : T'} \qquad \frac{\Gamma/\Delta \vdash a : \mathbf{global}\ T'[] \qquad \Gamma/\Delta \vdash e : \mathbf{TID}(a, b)}{\Gamma/(a', (a, b)), \Delta \vdash a[e] : T'}$$

$$\frac{\Gamma/\Delta \vdash e_1 : \text{uint}}{\Gamma/\Delta \vdash e_1 : \text{int}} \qquad \frac{\Gamma/\Delta \vdash e_1 : \mathbf{const}\ \text{uint}}{\Gamma/\Delta \vdash e_1 : \mathbf{TID}(0, e_1)} \qquad \frac{\Gamma/\Delta \vdash e_1 : \mathbf{const}\ T'}{\Gamma/\Delta \vdash e_1 : T'}$$

2.2 Commands

$$\frac{x{:}T, \Gamma/\Delta_r/\Delta_w \vdash S \text{ ok}}{\Gamma/\Delta_r/\Delta_w \vdash \textbf{decl } T\ x \textbf{ in } S \text{ ok}} \qquad \frac{x{:}T \in \Gamma \qquad T\texttt{scalar} \qquad \Gamma/\Delta_r \vdash e{:}T}{\Gamma/\Delta_r/\cdot \vdash x := e \text{ ok}} \qquad \frac{}{\Gamma/\cdot/\cdot \vdash \textbf{sync} \text{ ok}}$$

$$\frac{\Gamma/\Delta_r^1 \vdash e{:}\text{bool} \qquad \Gamma/\Delta_r^2/\Delta_w^2 \vdash S_1 \text{ ok} \qquad \Gamma/\Delta_r^3/\Delta_w^3 \vdash S_2 \text{ ok}}{\Gamma/\Delta_r^1 \cup \Delta_r^2 \cup \Delta_r^3/\Delta_w^2 \cup \Delta_w^3 \vdash \textbf{if } e \textbf{ then } S_1 \textbf{ else } S_2 \text{ ok}}$$

$$\frac{x{:}T \in \Gamma \qquad T = \textbf{global } T'[] \qquad \Gamma/\Delta_r^1 \vdash e_1{:}\text{uint} \qquad \Gamma/\Delta_r^2 \vdash e_2{:}T'}{\Gamma/\Delta_r^1 \cup \Delta_r^2/\cdot \vdash a[e_1] := e_2 \text{ ok}}$$

$$\frac{x{:}T \in \Gamma \qquad T = \textbf{global } T'[] \qquad \Gamma/\Delta_r^1 \vdash e_1{:}\textbf{TID}(a', b') \qquad \Gamma/\Delta_r^2 \vdash e_2{:}T'}{\Gamma/\Delta_r^1 \cup \Delta_r^2/(a, (a', b')) \vdash a[e_1] := e_2 \text{ ok}}$$

$$\frac{\begin{array}{ccccc} & & \Gamma/\delta_r^1 \vdash e_1{:}\text{int} & \Gamma/\delta_r^2 \vdash e_2{:}\text{int} & \\ \Gamma/\delta_r^3 \vdash e_3{:}\text{int} & \Gamma/\Delta_r^0/\Delta_w^0 \vdash S[e_1/x] \text{ ok} & \Gamma/\Delta_r^1/\Delta_w^1 \vdash S[e_1 + e_3/x] \text{ ok} & \dots & \Gamma/\Delta_r^{15}/\Delta_w^{15} \vdash S[e_1 + 15 * e_3/x] \text{ ok} \end{array}}{\Gamma/\bigcup \Delta_r \cup \bigcup \delta_r/\bigcup \Delta_w \vdash \textbf{for } x \textbf{ in } [e_1, e_2, e_3]\{S\} \text{ ok}}$$

The rule for the **for** essentially unrolls the first 16 iterations of the loop. If the loop variable is being used in an array index this rule collects the first 16 values that the index can take. The reason is that the same behaviour repeats itself as far as memory coalescing is concerned after the first 16 iterations as the difference in addresses is a multiple of 16 (and there are 16 threads in a half warp)

2.3 Statements

$$\frac{\Gamma/\Delta_r^1/\Delta_w^1 \vdash C \text{ ok} \qquad \Gamma/\Delta_r^2/\Delta_w^2 \vdash S \text{ ok}}{\Gamma/\Delta_r^1 \cup \Delta_r^2/\Delta_w^1 \cup \Delta_w^2 \vdash C; S \text{ ok}} \qquad \frac{}{\Gamma/\cdot/\cdot \vdash \langle\rangle \text{ ok}}$$