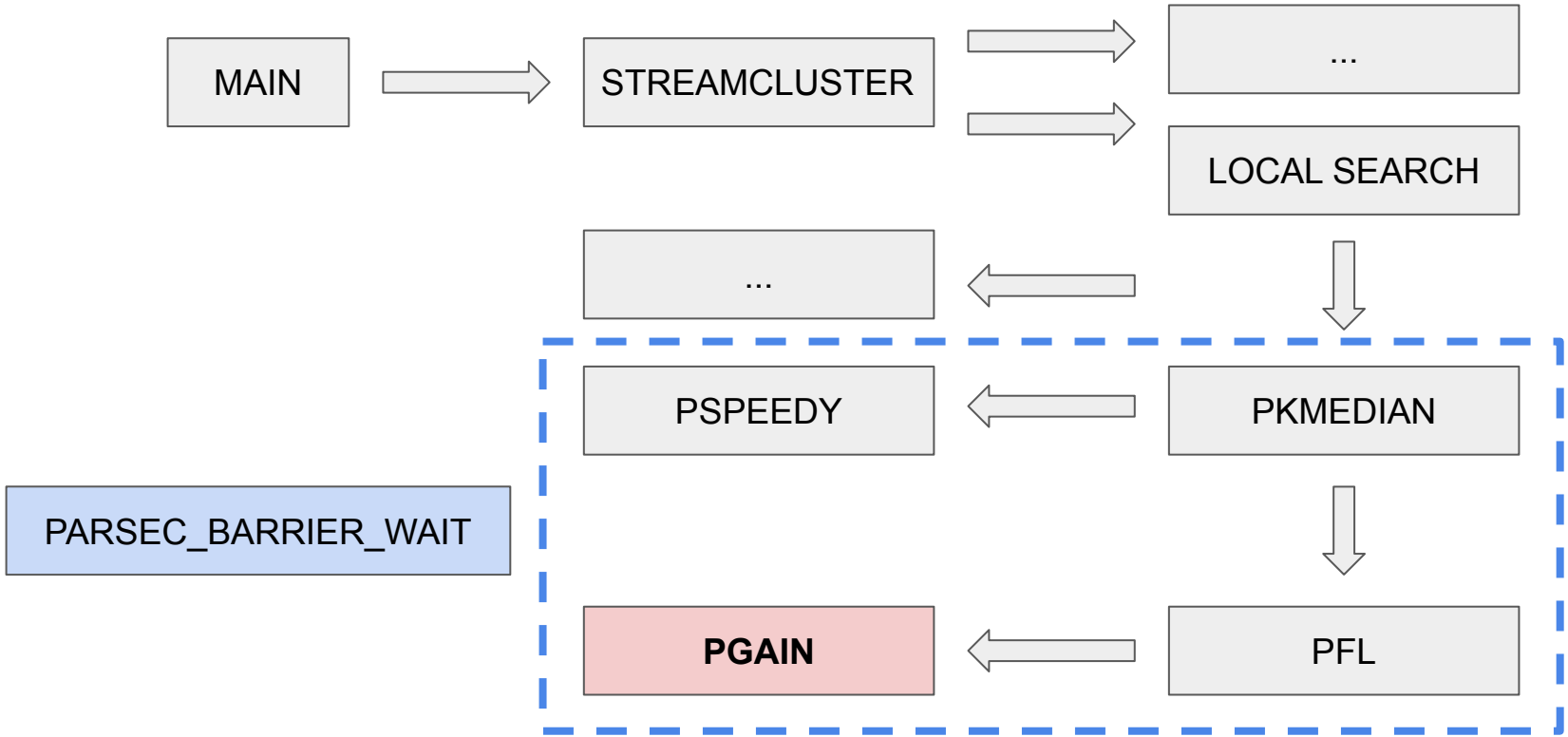
Análise de vetorização: Streamcluster

Alunos: Ângelo Leite, Ângelo Marcelino, Jhonatan Heberson, Maurício Lima

Estruturas de regiões paralelas e laços de aplicação



Estruturas de regiões paralelas e laços de aplicação

- **pgain**: 9 laços não-vetorizados, 9 barreiras de *thread*
  - calcula constantemente ganho pela possível criação de novo centróide;
- **pspeedy**: 7 laços não-vetorizados, 9 barreiras de *thread*
  - calcula custo da solução pelo cálculo de distância entre pontos;
- **pkmedian**: 4 laços não-vetorizados, 4 barreiras de *thread*
  - calcula centro mediano aos pontos;
- **pLF**: 2 laços não-vetorizados, 3 barreiras de *thread*
  - calcula custo da nova localização

Detalhes das flags usadas para auto-vetorizar os laços

-O3 -fopt-info-vec-all - **para imprimir todas as informações**
-O3 -fopt-info-vec-missed - **para imprimir apenas os laços não otimizados**
-O3 -fopt-info-vec-optimized - **para imprimir apenas os laços otimizados**

Comandos de execução

/usr/bin/g++ **-O3 -fopt-info-vec-all** -g -funroll-loops -fprefetch-loop-arrays -fpermissive -fno-exceptions -static-libgcc -Wl,--hash-style=both,--as-needed -DPARSEC_VERSION=3.0-beta-20150206 -DENABLE_THREADS -pthread -c streamcluster.cpp

/usr/bin/g++ **-O3 -fopt-info-vec-all** -g -funroll-loops -fprefetch-loop-arrays -fpermissive -fno-exceptions -static-libgcc -Wl,--hash-style=both,--as-needed -DPARSEC_VERSION=3.0-beta-20150206 -DENABLE_THREADS -pthread -c parsec_barrier.cpp

/usr/bin/g++ **-O3 -fopt-info-vec-all** -g -funroll-loops -fprefetch-loop-arrays -fpermissive -fno-exceptions -static-libgcc -Wl,--hash-style=both,--as-needed -DPARSEC_VERSION=3.0-beta-20150206 -DENABLE_THREADS -pthread -L/usr/lib64 -L/usr/lib streamcluster.o parsec_barrier.o -o streamcluster

Flag `-O_` - Indica tipo/objetivo de otimização

Table 3-8 Optimization goals

Optimization goal	Useful optimization levels
Smaller code size	`-Oz`, `-Omin`
Faster performance	`-O2`, `-O3`, `-Ofast`, `-Omax`
Good debug experience without code bloat	`-O1`
Better correlation between source code and generated code	`-O0` (no optimization)
Faster compile and build time	`-O0` (no optimization)
Balanced code size reduction and fast performance	`-Os`

- As opções `-O2` e `-O3` são as opções que procuram adicionar vetorização ao compilar o código.
- No `GCC`, `-O3` habilita flags de otimização:

```
-ftree-loop-vectorize

-ftree-slp-vectorize
```

Estudo de estratégias para vetorização

Problemas encontrados a partir da análise de ocorrências nas principais funções da aplicação

1. streamcluster.cpp:1000:22: missed: couldn't vectorize loop
   streamcluster.cpp:1000:22: missed: not vectorized: control flow in loop.

2. streamcluster.cpp:1109:23: missed: couldn't vectorize loop
   streamcluster.cpp:1111:28: missed: not vectorized: relevant stmt not supported: _154 = _152 + gl_cost_of_opening_x_lsm.541_227;

3. streamcluster.cpp:1083:25: missed: couldn't vectorize loop
   streamcluster.cpp:1083:25: missed: Loop costings may not be worthwhile.

4. parsec_barrier.cpp:156:21: missed: not vectorized: volatile type: _8 ={v}

5. parsec_barrier.cpp:185:57: missed: couldn't vectorize loop
   parsec_barrier.cpp:185:57: missed: not vectorized: number of iterations cannot be computed.

6. parsec_barrier.cpp:192:31: missed: statement clobbers memory: pthread_mutex_unlock (_1);

1. Fluxo de controle dentro do loop: quando uma ou mais condições precisam ser satisfeitas para que se realize uma operação. Ex.:

```
for( int i = k1; i < k2; i++ )
    if( is_center[i] && gl_lower[center_table[i]] > 0 )
      is_center[i] = false;
```

2. Declaração não suportada: operações que impossibilitem vetorização, como acesso a múltiplas variáveis não sequenciais independentes. Ex.:

```
for( int p = 0; p < nproc; p++ ) {
    gl_number_of_centers_to_close += (int)work_mem[p*stride + K];
    gl_cost_of_opening_x += work_mem[p*stride+K+1];
  }
```

3. Custo não compensa: o compilador assume que haverá poucas iterações no laço, logo não autoriza vetorização do mesmo. Ex.:

```
for( int p = 0; p < nproc; p++ )
        low += work_mem[center_table[i]+p*stride];
```

4. Variáveis do tipo volátil: variáveis voláteis são aquelas que podem mudar a qualquer momento, imprevisivelmente, por alguma thread ou processo. O compilador evita suposições e otimizações quando se trata de voláteis. Ex.:

```
volatile spin_counter_t i=0;
while(barrier->is_arrival_phase && i<SPIN_COUNTER_MAX) i++;
```

5. Número de iterações não pôde ser computado: quando o compilador não consegue identificar ou assumir um valor para o número de iterações no laço. Ex.:

```
while((rv=pthread_mutex_trylock(&barrier->mutex)) == EBUSY);
```

6. Presença de "memory clobbers": quando o acesso a memória sequencial ocorre de forma inconsistente. Ex.:

```
while(barrier->is_arrival_phase) {
      rv = pthread_cond_wait(&barrier->cond, &barrier->mutex);
      if(rv != 0) {
        pthread_mutex_unlock(&barrier->mutex);
        return rv;
      }
```