prefetcher.hpp

Alterar esta estrutura para:

```
-   std::vector<uint64_t> prefetch_waiting_complete;
+   std::vector<memory_package_t *> prefetched_lines;
```

(precisamos manter um rastreio de quando a requisição está pronta)

prefetcher.cpp - prefetch

Verificação dos prefetches completos

```
// ******************************
// Remove um prefetch já completo
// ******************************

if ((this->prefetched_lines.size() != 0) &&
    (this->prefetched_lines.front().readyAt <= orcs_engine.get_global_cycle()) &&
    (this->prefetched_lines.front().status == PACKAGE_STATE_WAIT))
{
  /* Insere na cache recebida */
  line_t *linha = cache->installLine(request, latency_prefetch, idx_padding, line_padding);

  /* Marca como prefetched */
  linha->prefetched=1;

  /* Remove esse mais antigo que já está completo */
  delete this->prefetched_lines.begin();
  this->prefetched_lines.erase(this->prefetched_lines.begin());
}
```

Impede mais requisições de prefetch que o limite

```
if (this->prefetched_lines.size() >= PARALLEL_PREFETCH) {
  return;
}
```

Criação de uma nova entrada de prefetch

```
memory_package_t *pk = new memory_package_t();

this->prefetched_lines.push_back(pk);
```

Criação do pacote DRAM

```
pk->opcode_address = 0x0;
pk->memory_address = newAddress;
pk->memory_size = mob_line->memory_size;
pk->memory_operation = mob_line->memory_operation;
pk->status = PACKAGE_STATE_UNTREATED;
pk->is_hive = false;
pk->is_vima = false;
pk->hive_read1 = mob_line->hive_read1;
pk->hive_read2 = mob_line->hive_read2;
pk->hive_write = mob_line->hive_write;
pk->readyAt = orcs_engine.get_global_cycle();
pk->born_cycle = orcs_engine.get_global_cycle();
pk->sent_to_ram = false;
pk->type = DATA;
pk->uop_number = 0;
pk->processor_id = this->processor_id;
pk->op_count[request->memory_operation]++;
pk->clients.shrink_to_fit();
```

Envio para a DRAM

```
orcs_engine.memory_controller->add_requests_prefetcher();
orcs_engine.memory_controller->requestDRAM(pk);
```