NOPL & Compiler to NOP C++ namespace multi-threading oriented solution: studies of capabilities for the x86-64 architecture

Guilherme Martini

Adriano Ronszcka

Jean Simão

João Fabro

Objetivo do trabalho

- Estudar a viabilidade de rodar códigos PON em arquiteturas PC multicores usando recursos de multithreading.

Definição das etapas principais

1. Conhecer a estrutura do codigo Cpp gerado a partir de um codigo NOP e decidir como a divisão de tarefas em múltiplos núcleos serão feitas.

2. Programar uma versão que use Pthreads e Thread Pooling manualmente.

3. Modificar o código do compilador NOP 1.0 para gerar o mesmo código automaticamente. Eles devem ser idênticos.

4. Manualmente, adicionar mais carga computacional no programa

5. Mostrar resultados e limitações.

Etapa 1 – Estudo do código e definição de ponto de Multithreading

- A partir da observação do código gerado (o código do portão eletrônico foi usado como referência), foi decidido criar pontos de multithreading na chamadas dos métodos (após a satisfação de premissas).

- O fator determinante para esta decisão foi o fato de que as outras entidades PON seriam somente dados ou condicionais, que no caso não exerceriam uso de muitas instruções para serem executadas.

Etapa 1 – Estudo do código e definição de ponto de Multithreading

- Exemplo de adição de Thread Pooling ao arquivo “premisses.cpp”:

```
namespace prRemoteControlOn{
    bool state = false;
    int cpy1st, cpy2nd;
    threadpool thpool1, thpool2;
    void init(){
        cpy1st = 0;
        cpy2nd = 1;
        thpool1 = thpool_init(2);
        thpool2 = thpool_init(2);
        if(cpy1st  ==  cpy2nd){
            state = true;
            thpool_add_work(thpool1, subCondition::a1::inc, NULL);
            thpool_add_work(thpool1, subCondition::a2::inc, NULL);
        }
    }
```

```
void inc(void* arg){
    count++;
    if (count == 2){
        instantiation::event::mt::reset();
        instantiation::gate::mt::opened();
        crc32_many_times();
    }
}
```

Etapa 2 – Programar uma versão de Pthreads e ThreadPooling manualmente

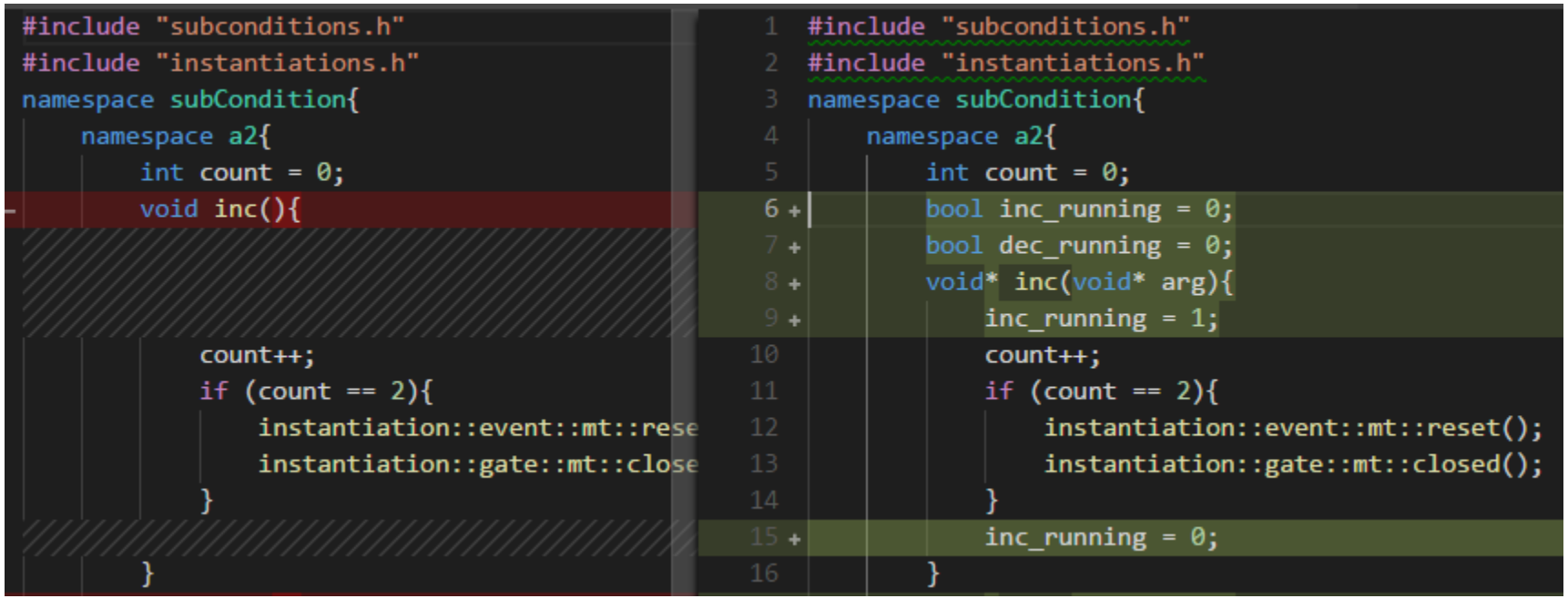
- A inserção manual destes recursos foram feitos sobre o CPP gerado pela versão de namespaces do compilador NOP 1.0. Abaixo as principais mudanças da versão com Pthreads:

```
namespace prGateIsOpened{
    bool state = false;
    int cpy1st, cpy2nd;

    void init(){
        cpy1st = 0;
        cpy2nd = 1;
        if(cpy1st  ==  cpy2nd){
            state = true;
            subCondition::a2::inc();


        }
    }
```

```
44      namespace prGateIsOpened{
45          bool state = false;
46          int cpy1st, cpy2nd;
47  +       pthread_t id1, id2;
48          void init(){
49              cpy1st = 0;
50              cpy2nd = 1;
51              if(cpy1st  ==  cpy2nd){
52                  state = true;

53
54  +               while(subCondition::a2::inc_running == 1) {};
55  +               pthread_create(&id1, NULL, &subCondition::a2::inc, NULL);
56              }
57          }
```

Etapa 2 – Programar uma versão de Pthreads e ThreadPooling manualmente

- Principais mudanças, parte 2, Pthreads:



```
#include "subconditions.h"
#include "instantiations.h"
namespace subCondition{
    namespace a2{
        int count = 0;
        void inc(){

            count++;
            if (count == 2){
                instantiation::event::mt::rese
                instantiation::gate::mt::close
            }

        }
```

```
1  #include "subconditions.h"
2  #include "instantiations.h"
3  namespace subCondition{
4      namespace a2{
5          int count = 0;
6  +       bool inc_running = 0;
7  +       bool dec_running = 0;
8  +       void* inc(void* arg){
9  +           inc_running = 1;
10             count++;
11             if (count == 2){
12                 instantiation::event::mt::reset();
13                 instantiation::gate::mt::closed();
14             }
15 +           inc_running = 0;
16         }
```

Etapa 2 – Programar uma versão de Pthreads e ThreadPooling manualmente

- Principais mudanças, Thread Pooling:

```
    namespace prRemoteControlOn{
        bool state = false;
        int cpy1st, cpy2nd;

        void init(){
            cpy1st = 0;
            cpy2nd = 1;


            if(cpy1st  ==  cpy2nd){
                state = true;
                subCondition::a1::inc();
subCondition::a2::inc();


            }
        }
```

```
75      namespace prRemoteControlOn{
76          bool state = false;
77          int cpy1st, cpy2nd;
78  +       threadpool thpool1, thpool2;
79          void init(){
80              cpy1st = 0;
81              cpy2nd = 1;
82  +           thpool1 = thpool_init(2);
83  +           thpool2 = thpool_init(2);
84              if(cpy1st  ==  cpy2nd){
85                  state = true;


86
87  +               thpool_add_work(thpool1, subCondition::a1::inc, NULL);
88  +               thpool_add_work(thpool1, subCondition::a2::inc, NULL);
89  +               thpool_wait(thpool1);
90              }
91          }
```

Etapa 2 – Programar uma versão de Pthreads e ThreadPooling manualmente

- Principais mudanças, parte 2, Thread Pooling:

```
#include "subconditions.h"
#include "instantiations.h"
namespace subCondition{
    namespace a2{
        int count = 0;
        void inc(){
            count++;
            if (count == 2){
                instantiation::event::mt::reset();
                instantiation::gate::mt::closed();
            }
        }
```

```
1  #include "subconditions.h"
2  #include "instantiations.h"
3  namespace subCondition{
4      namespace a2{
5          int count = 0;
6          void inc(void* arg){
7              count++;
8              if (count == 2){
9                  instantiation::event::mt::reset();
10                 instantiation::gate::mt::closed();
11             }
12         }
```

Etapa 3 – Modificar o código do compilador NOP 1.0 para gerar o mesmo código automaticamente.

- Após as versões manuais compilarem e funcionarem da mesma maneira que a versão original, foi modificado o compilador NOP v1.0 sobre a versões Namespaces para que este pudesse gerar código com Pthreads e Thread Pooling automaticamente. Arquivo bison_pon.y:

```
-   }else if (atoi(argv[1]) == 6) {
        compiler = new StaticCPPCompiler();
        cout << "Compilado para StacticC++, veja os resultados na pasta cp
-   }else if (atoi(argv[1]) == 7) {
    compiler = new VHDLCompiler();
    cout << "Compilado para VHDL, veja os resultados na pasta VHDLcompilad
-  }else if (atoi(argv[1]) == 8) {
    compiler = new NPCompiler();
    cout << "Compilado com o uso de namespaces, veja os resultados na past
-  }else {
        cout << "Escolha uma ação correta:" << endl;
        cout << "1 - C" << endl;
        cout << "2 - C++" << endl;
        cout << "3 - framework" << endl;
        cout << "4 - NOP" << endl;
        cout << "5 - NOCA" << endl;
        cout << "6 - Static C++" << endl;
        cout << "7 - VHDL" << endl;
        cout << "8 - Namespace C++" << endl;
        exit(0);
    }
```

```
466 +   } else if (atoi(argv[1]) == 6) {
467         compiler = new StaticCPPCompiler();
468         cout << "Compilado para StacticC++, veja os resultados na pasta cppcompilados" << endl;
469 +   } else if (atoi(argv[1]) == 7) {
470     compiler = new VHDLCompiler();
471     cout << "Compilado para VHDL, veja os resultados na pasta VHDLcompilados" << endl;
472 +   } else if (atoi(argv[1]) == 8) {
473         compiler = new NPCompiler();
474         cout << "Compilado com o uso de namespaces, veja os resultados na pasta cppcompilados" << endl;
475 +   } else if (atoi(argv[1]) == 9) {
476 +       compiler = new NPCompiler_PThreads();
477 +       cout << "Compilado com o uso de namespaces com Pthreads, veja os resultados na pasta cppcompilados" << endl;
478 +   } else if (atoi(argv[1]) == 10) {
479 +       compiler = new NPCompiler_TPool();
480 +       cout << "Compilado com o uso de namespaces com Thread Pooling, veja os resultados na pasta cppcompilados" << endl;
481 +   } else {
482         cout << "Escolha uma ação correta:" << endl;
483         cout << "1 - C" << endl;
484         cout << "2 - C++" << endl;
485         cout << "3 - framework" << endl;
486         cout << "4 - NOP" << endl;
487         cout << "5 - NOCA" << endl;
488         cout << "6 - Static C++" << endl;
489         cout << "7 - VHDL" << endl;
490         cout << "8 - Namespace C++" << endl;
491 +       cout << "9 - Namespace C++ com Pthreads" << endl;
492 +       cout << "10 - Namespace C++ com Thread Pooling" << endl;
493         exit(0);
494     }
```

Etapa 3 – Modificar o código do compilador NOP 1.0 para gerar o mesmo código automaticamente.

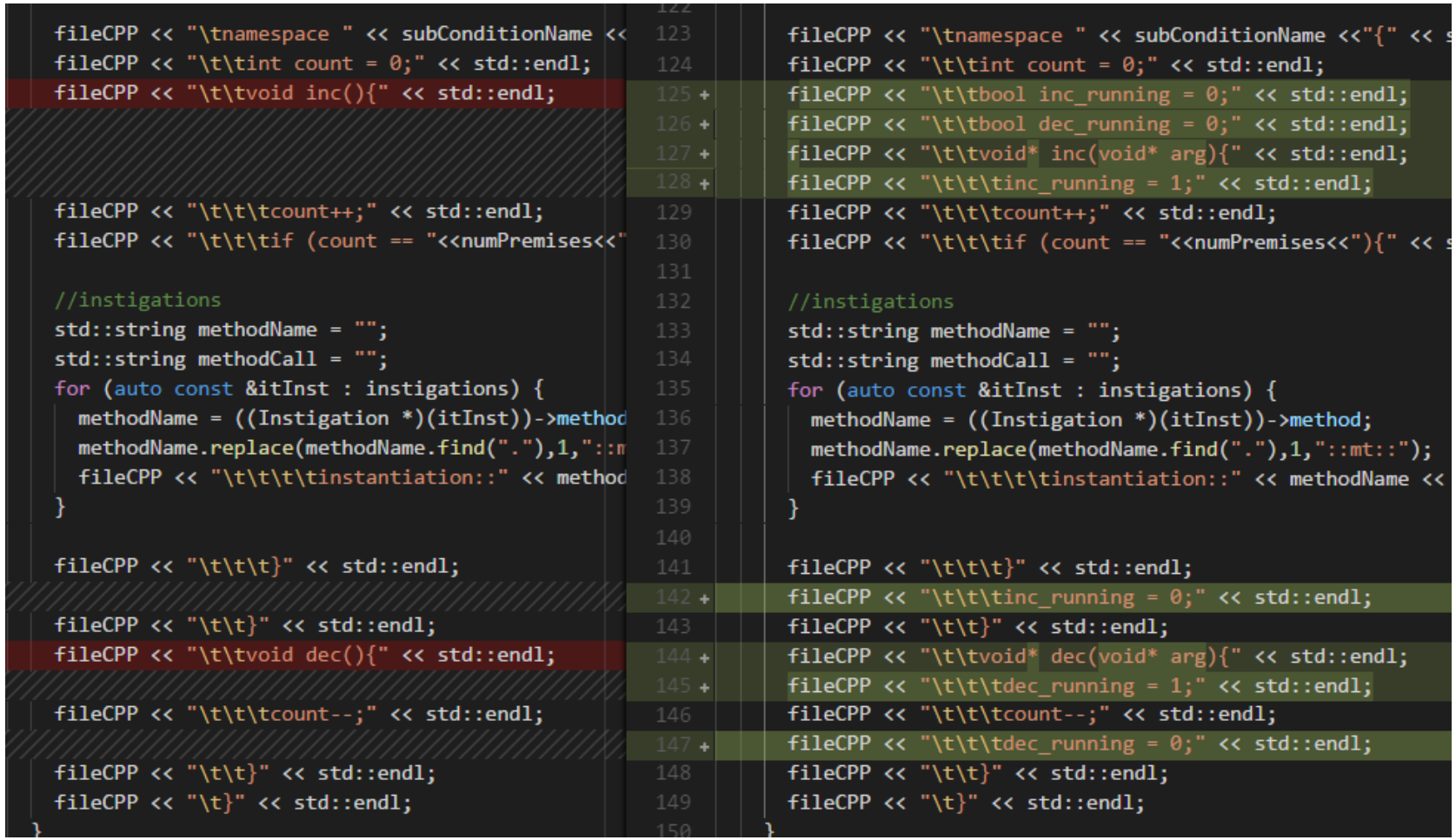
- Arquivo NPCompilerPthreads.cpp :

```
std::list<SubCondition *> subconditions = (it.second)->subconditions;
for (auto const &itCond : subconditions) {
  methodCallInc += "subCondition::" + ((SubCondition *)(itCond))->userEntityId + ":
  methodCallDec += "subCondition::" + ((SubCondition *)(itCond))->userEntityId + ":
}
```

```
188 +   int subConditionCounter = 0;
189     std::list<SubCondition *> subconditions = (it.second)->subconditions;
190     for (auto const &itCond : subconditions) {
191 +
192 +     if(methodCallInc.compare("") != 0) methodCallInc += "\t\t\t\t\t";
193 +     if(methodCallDec.compare("") != 0) methodCallDec += "\t\t\t\t\t";
194 +     methodCallInc += "while(subCondition::" + ((SubCondition *)(itCond))->userEntityId + "::i
195 +     methodCallDec += "while(subCondition::" + ((SubCondition *)(itCond))->userEntityId + "::c
196 +
197 +     if(methodCallInc.compare("") != 0) methodCallInc += "\t\t\t\t\t";
198 +     if(methodCallDec.compare("") != 0) methodCallDec += "\t\t\t\t\t";
199 +     methodCallInc += "pthread_create(&id" + std::to_string(subConditionCounter  ) + ", NULL,
200 +     methodCallDec += "pthread_create(&id" + std::to_string(subConditionCounter+1) + ", NULL,
201 +     subConditionCounter+=2;
202     }
```

Etapa 3 – Modificar o código do compilador NOP 1.0 para gerar o mesmo código automaticamente.

- Arquivo NPCompilerPthreads.cpp, parte 2 :

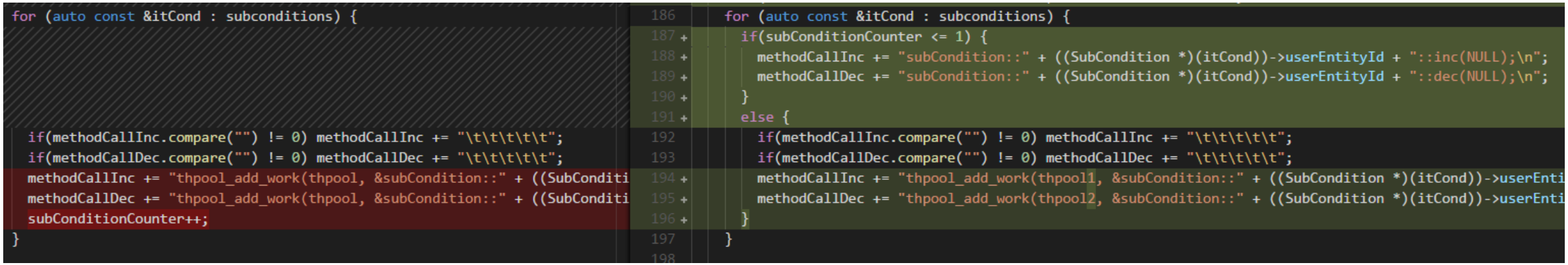


```
fileCPP << "\tnamespace " << subConditionName <<
fileCPP << "\t\tint count = 0;" << std::endl;
fileCPP << "\t\tvoid inc(){" << std::endl;

fileCPP << "\t\t\tcount++;" << std::endl;
fileCPP << "\t\t\tif (count == "<<numPremises<<"

//instigations
std::string methodName = "";
std::string methodCall = "";
for (auto const &itInst : instigations) {
  methodName = ((Instigation *)(itInst))->method
  methodName.replace(methodName.find("."),1,"::m
  fileCPP << "\t\t\t\tinstantiation::" << method
}

fileCPP << "\t\t\t}" << std::endl;

fileCPP << "\t\t}" << std::endl;
fileCPP << "\t\tvoid dec(){" << std::endl;

fileCPP << "\t\t\tcount--;" << std::endl;

fileCPP << "\t\t}" << std::endl;
fileCPP << "\t}" << std::endl;
}
```

```
123   fileCPP << "\tnamespace " << subConditionName <<"{" << s
124   fileCPP << "\t\tint count = 0;" << std::endl;
125 + fileCPP << "\t\tbool inc_running = 0;" << std::endl;
126 + fileCPP << "\t\tbool dec_running = 0;" << std::endl;
127 + fileCPP << "\t\tvoid* inc(void* arg){" << std::endl;
128 + fileCPP << "\t\t\tinc_running = 1;" << std::endl;
129   fileCPP << "\t\t\tcount++;" << std::endl;
130   fileCPP << "\t\t\tif (count == "<<numPremises<<"){" << s
131
132   //instigations
133   std::string methodName = "";
134   std::string methodCall = "";
135   for (auto const &itInst : instigations) {
136     methodName = ((Instigation *)(itInst))->method;
137     methodName.replace(methodName.find("."),1,"::mt::");
138     fileCPP << "\t\t\t\tinstantiation::" << methodName <<
139   }
140
141   fileCPP << "\t\t\t}" << std::endl;
142 + fileCPP << "\t\t\tinc_running = 0;" << std::endl;
143   fileCPP << "\t\t}" << std::endl;
144 + fileCPP << "\t\tvoid* dec(void* arg){" << std::endl;
145 + fileCPP << "\t\t\tdec_running = 1;" << std::endl;
146   fileCPP << "\t\t\tcount--;" << std::endl;
147 + fileCPP << "\t\t\tdec_running = 0;" << std::endl;
148   fileCPP << "\t\t}" << std::endl;
149   fileCPP << "\t}" << std::endl;
150 }
```

Etapa 3 – Modificar o código do compilador NOP 1.0 para gerar o mesmo código automaticamente.

- Arquivo NPCompilerTPool.cpp:

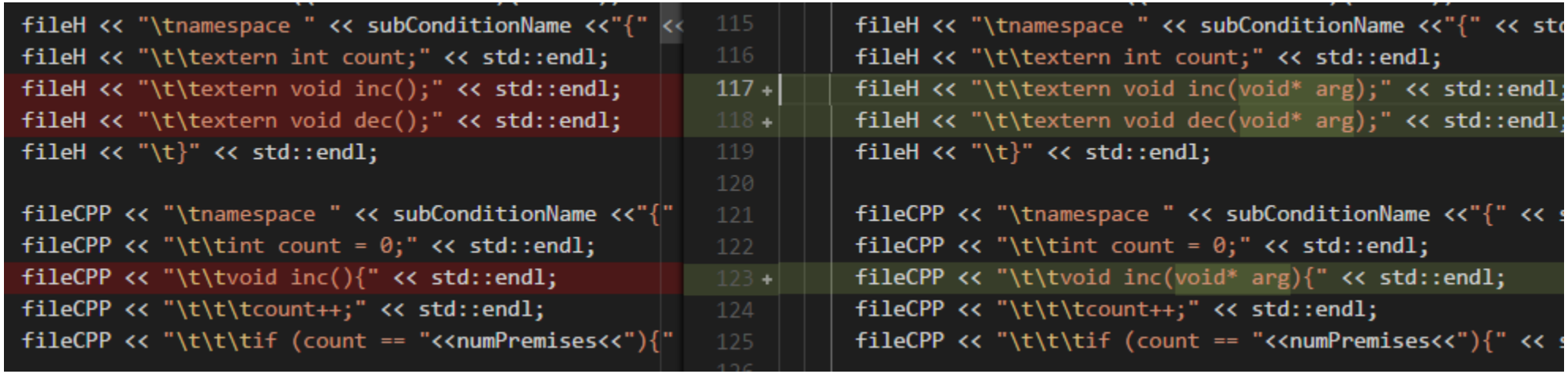


```
for (auto const &itCond : subconditions) {
  if(methodCallInc.compare("") != 0) methodCallInc += "\t\t\t\t\t";
  if(methodCallDec.compare("") != 0) methodCallDec += "\t\t\t\t\t";
  methodCallInc += "thpool_add_work(thpool, &subCondition::" + ((SubConditi
  methodCallDec += "thpool_add_work(thpool, &subCondition::" + ((SubConditi
  subConditionCounter++;
}
```

```
186     for (auto const &itCond : subconditions) {
187 +     if(subConditionCounter <= 1) {
188 +       methodCallInc += "subCondition::" + ((SubCondition *)(itCond))->userEntityId + "::inc(NULL);\n";
189 +       methodCallDec += "subCondition::" + ((SubCondition *)(itCond))->userEntityId + "::dec(NULL);\n";
190 +     }
191 +     else {
192         if(methodCallInc.compare("") != 0) methodCallInc += "\t\t\t\t\t";
193         if(methodCallDec.compare("") != 0) methodCallDec += "\t\t\t\t\t";
194 +       methodCallInc += "thpool_add_work(thpool1, &subCondition::" + ((SubCondition *)(itCond))->userEnti
195 +       methodCallDec += "thpool_add_work(thpool2, &subCondition::" + ((SubCondition *)(itCond))->userEnti
196 +     }
197     }
```

Etapa 3 – Modificar o código do compilador NOP 1.0 para gerar o mesmo código automaticamente.

- Arquivo NPCompilerTPool.cpp, parte 2:



Etapa 4 – Manualmente, adicionar mais carga computacional no programa

- Após a garantia que o código gerado pelo compilador NOP era idêntico ao gerado manualmente, foi adicionado ao CPP gerado de todas as versões (original-Namespaces, Pthreads e ThreadPooling) um novo método para aumentar a carga computacional do programa, simulando um programa NOP mais carregado computacionalmente.

- A função escolhida foi a geração de um CRC32 (Cyclic Redundancy Check 32bits) sobre um vetor de 50 bytes. A operação do CRC32 resulta em 100 XORs e 50 rotações a direita para cada chamada.

- A cada satisfação de premissa, o CRC32 é chamado entre 3mil e 3 milhões de vezes, aumentando em 450mil e 450 milhões de operações por satisfação de premissa. Este algoritmo é não otimizável.

Etapa 4 – Manualmente, adicionar mais carga computacional no programa.

- Exemplo do CRC32:

```
// calculate a checksum on a buffer -- start address = p, length = bytelength
unsigned int crc32_byte(unsigned char *p, unsigned int bytelength)
{
    unsigned int crc = 0xffffffff;
    while (bytelength-- !=0) crc = poly8_lookup[((unsigned char) crc ^ *(p++))] ^ (crc >> 8);
    // return (~crc); also works
    return (crc ^ 0xffffffff);
}

void crc32_many_times(void)
{
for(int i=0; i<CRC32_N_LOOPS; i++) crc32_byte(buffer, 50);
}
```

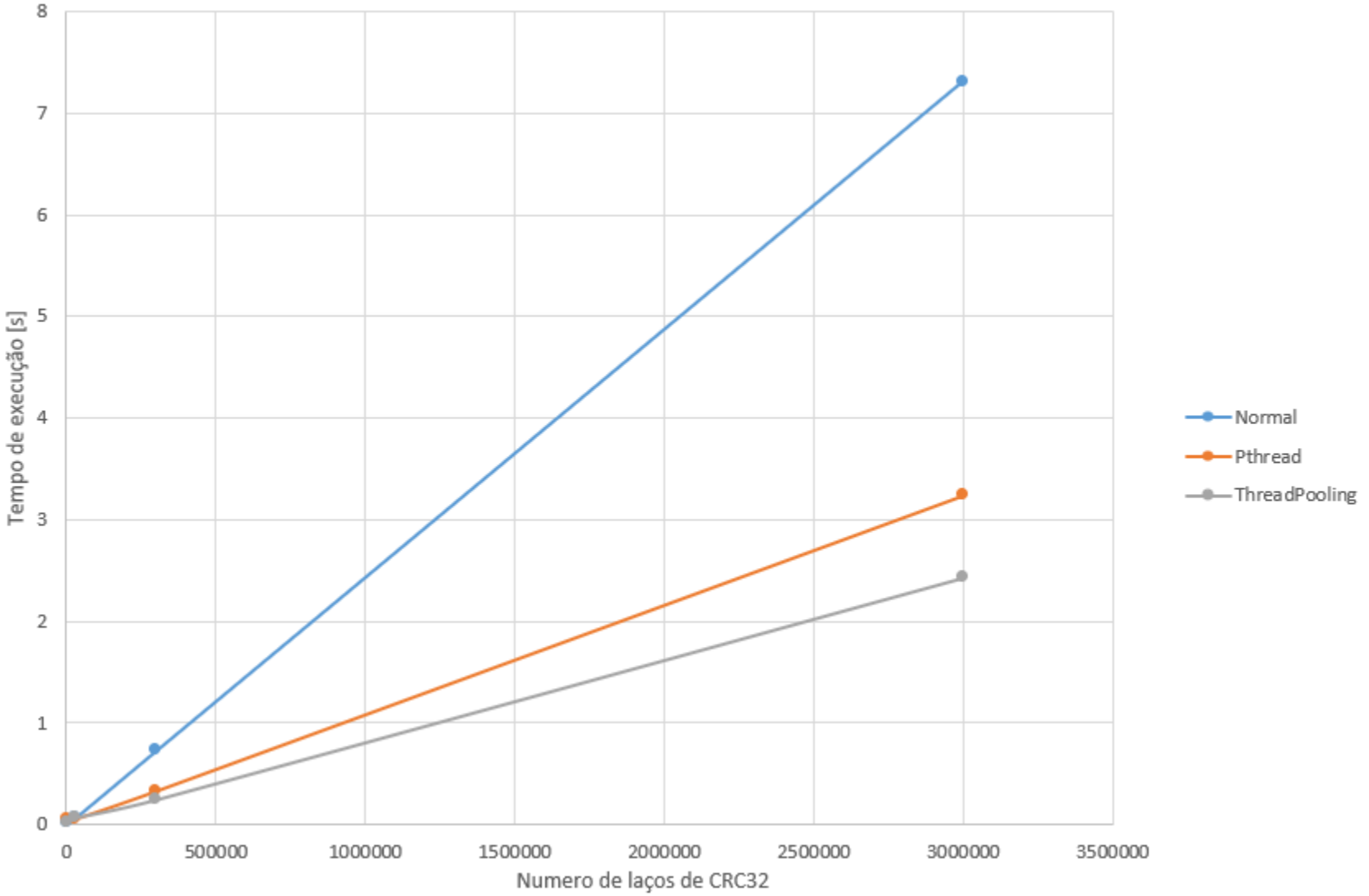
```
const unsigned int poly8_lookup[256] =
{
    0, 0x77073096, 0xEE0E612C, 0x990951BA,
    0x076DC419, 0x706AF48F, 0xE963A535, 0x9E6495A3,
    0x0EDB8832, 0x79DCB8A4, 0xE0D5E91E, 0x97D2D988,
    0x09B64C2B, 0x7EB17CBD, 0xE7B82D07, 0x90BF1D91,
    0x1DB71064, 0x6AB020F2, 0xF3B97148, 0x84BE41DE,
    0x1ADAD47D, 0x6DDDE4EB, 0xF4D4B551, 0x83D385C7,
    0x136C9856, 0x646BA8C0, 0xFD62F97A, 0x8A65C9EC,
    0x14015C4F, 0x63066CD9, 0xFA0F3D63, 0x8D080DF5,
    0x3B6E20C8, 0x4C69105E, 0xD56041E4, 0xA2677172,
```

Etapa 5 – Mostrar resultados e limitações

Normal 3000	Normal 30000	Normal 300000	Normal 3000000	
0.0156	0.0625	0.7344	7.3438	
0.0156	0.0625	0.7188	7.2969	
0.0156	0.0781	0.7344	7.2813	
0.0156	0.0781	0.7344	7.2969	
0.0156	0.0781	0.7344	7.2969	
0.015625	**0.071875**	**0.73125**	**7.303128**	**Media**
0	**0.008558165**	**0.006987712**	**0.023695497**	**DesvPad**

PThread 300(	PThread 30000	PThread 300000	PThread 3000000	
0.0469	0.0625	0.3282	3.2813	
0.0469	0.0469	0.3282	3.1719	
0.0469	0.0625	0.3282	3.1719	
0.0469	0.0469	0.3282	3.2813	
0.0469	0.0469	0.3282	3.2813	
0.046875	**0.053125**	**0.32815**	**3.237502**	**Media**
0	**0.008558165**	**0**	**0.059904416**	**DesvPad**

TPooling 300	TPooling 30000	TPooling 30000	TPooling 3000000	
0.0156	0.0625	0.2656	2.5781	
0.0156	0.0781	0.2688	2.5313	
0.0156	0.0781	0.2344	2.6094	
0.0156	0.0625	0.2688	1.8750	
0.0156	0.0781	0.2031	2.5313	
0.015625	**0.071875**	**0.248125**	**2.425**	**Media**
0	**0.008558165**	**0.02903056**	**0.309240943**	**DesvPad**



Etapa 5 – Mostrar resultados e limitações

Limitações no Pthread:

- Pelo fato dos métodos irem para as threads, a thread principal invoca um novo evento e recria uma thread que já estava com trabalho com um novo trabalho. Isso foi resolvido via sinaleiro, como mostrado no código dos slides anteriores.

- Os atributos eram atômicos o suficiente para não gerar conflitos de escrita, mas desconfia-se que para mudança dos mesmos será necessário uma estrutura de dados persistente.

Etapa 5 – Mostrar resultados e limitações

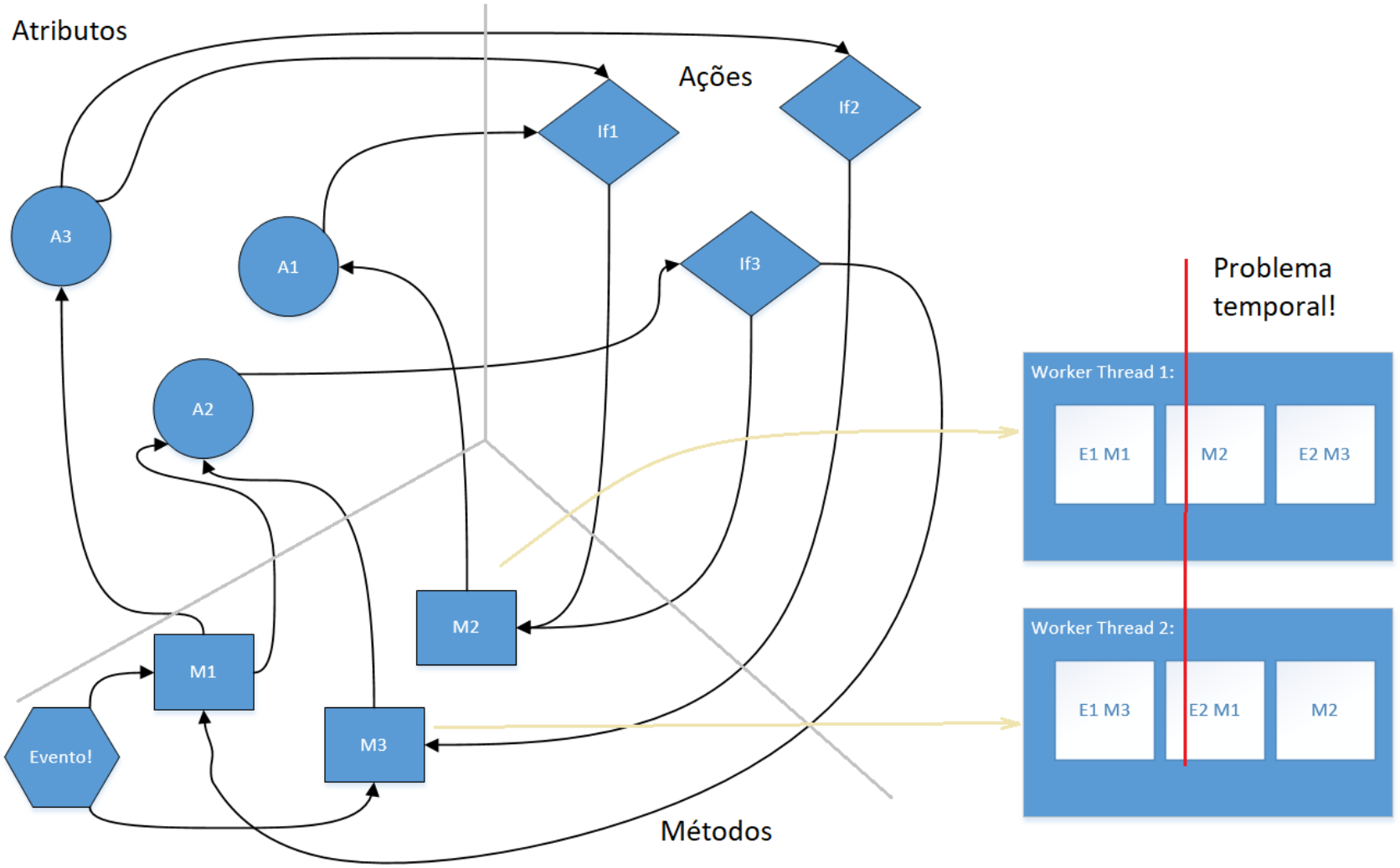
Limitações no ThreadPooling:

- Novos eventos geram ações que manipulam atributos antes de métodos já empilhados em worker threads, gerando problemas temporais de mudanças nos atributos (diferente do problema de prioridade entre métodos). Isso cria a necessidade adicionar um escalonamento mais inteligente de eventos e invocação de métodos.

- Para a versão do compilador NOP, foi gerado uma versão que possui uma garantia de fim de serviço (verificação de worker thread vazia antes de continuar). O resultado em tempo passa a ser equivalente ao de Pthreads.

- Na versão manual, mesmo com o problema de incoerência temporal, foi gerada uma versão para caso ótimo (sem sincronia de worker thread) que foi o exibido nos slides anteriores.

Etapa 5 – Mostrar resultados e limitações



Etapa 5 – Mostrar resultados e limitações

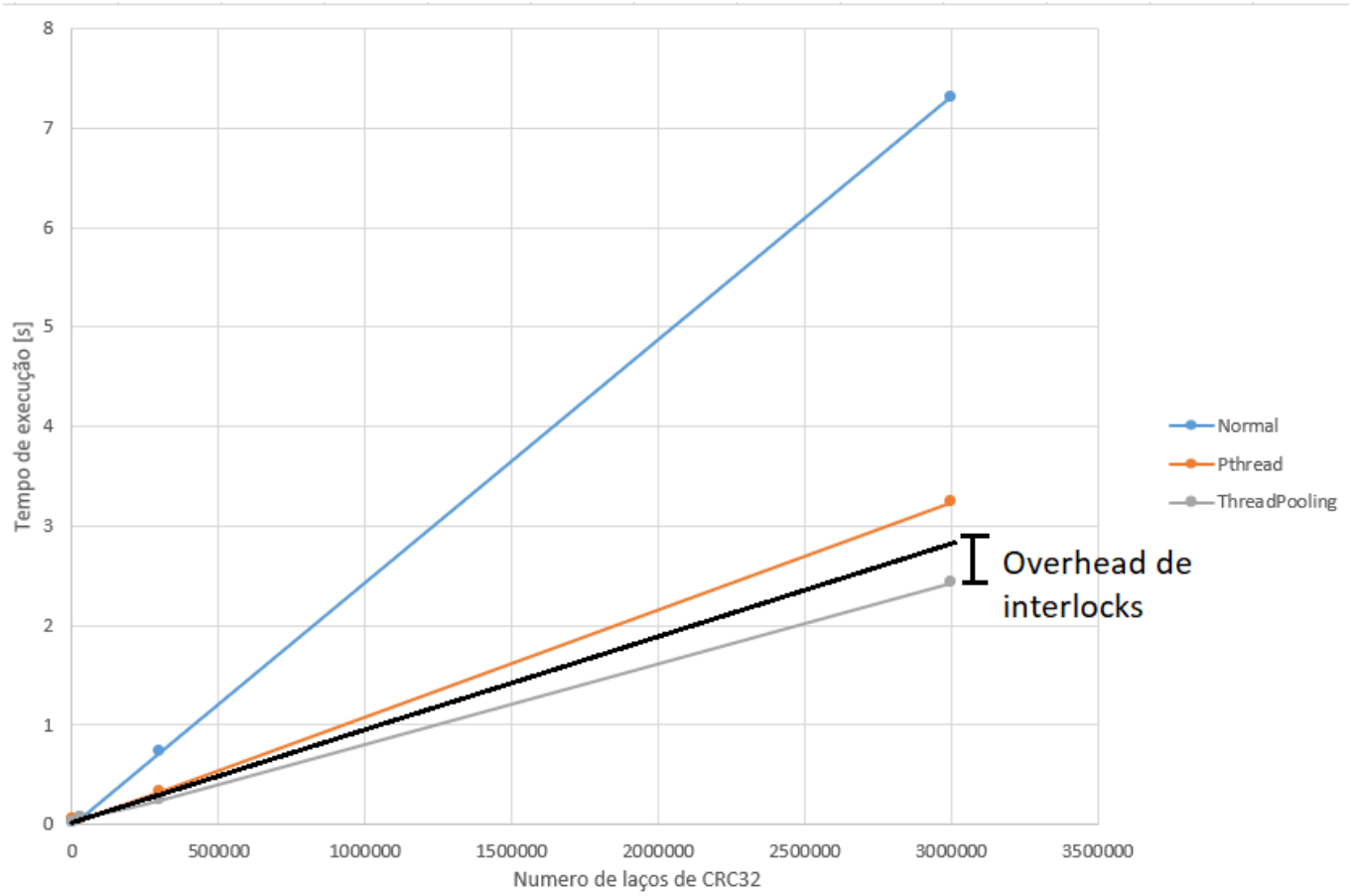
Garantia de fim de serviço no ThreadPooling:

```
int main() {

  clock_t begin = clock();

  premise::prGateIsClosed::init();
  premise::prGateIsOpened::init();
  premise::prRemoteControlOn::init();

  int iteration = 4;
  instantiation::gate::at::gateState::setValue(0);
  while(iteration--){
    instantiation::event::at::eventState::setValue(1);
  }

  clock_t end = clock();

  double time_spent = (double)(end - begin);
  time_spent /= ((double)CLOCKS_PER_SEC);
  cout << time_spent << endl;

  return 0;
}
```

```
10  int main() {
11
12    clock_t begin = clock();
13
14    premise::prGateIsClosed::init();
15    premise::prGateIsOpened::init();
16    premise::prRemoteControlOn::init();
17
18    int iteration = 4;
19    instantiation::gate::at::gateState::setValue(0);
20    while(iteration--){
21      instantiation::event::at::eventState::setValue(1);
22    }
23
24 +  thpool_wait(premise::prGateIsClosed::thpool1);
25 +  thpool_wait(premise::prGateIsOpened::thpool1);
26 +  thpool_wait(premise::prRemoteControlOn::thpool1);
27 +  thpool_wait(premise::prRemoteControlOn::thpool2);
28 +
29    clock_t end = clock();
30
31    double time_spent = (double)(end - begin);
32    time_spent /= ((double)CLOCKS_PER_SEC);
33    cout << time_spent << endl;
34
35    return 0;
36  }
```

Etapa 5 – Mostrar resultados e limitações

Provável resultado com adição de controle de escaloamento e persistência de dados:



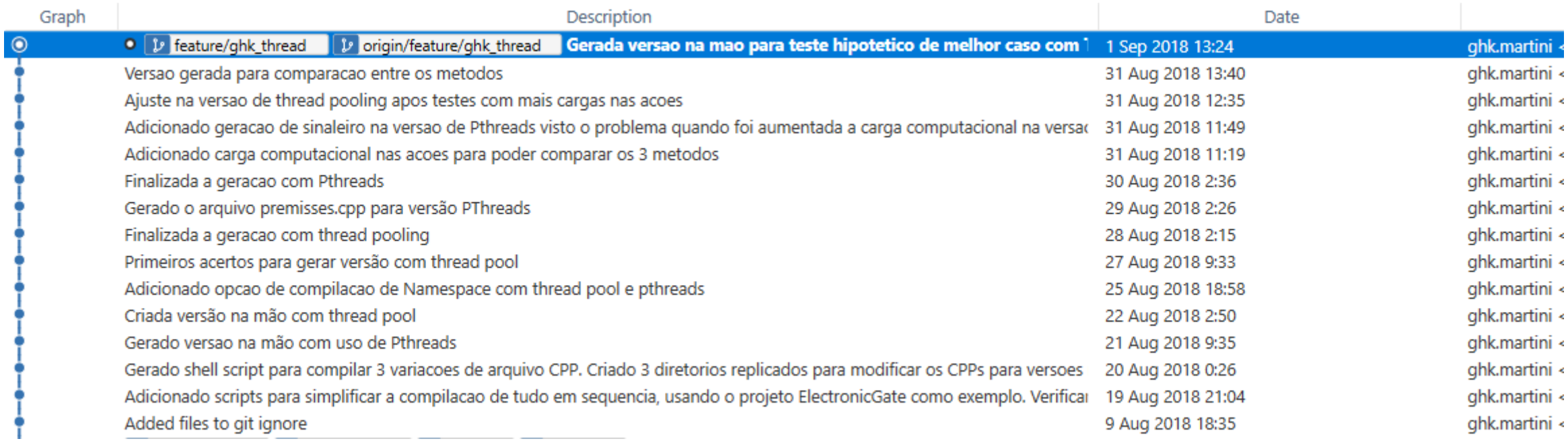
Etapas futuras

- Estudar se o problema de atributos que são estruturas de dados maiores sofreriam um problema de concorrência (quando a mudança do atributo é maior que uma instrução – não atômica). Um candidato para pesquisas é o Immutable++: https://github.com/rsms/immutable-cpp

- Estudar um modelo de broker para resolver o escalonamento entre worker threads. Olhar se o estudo do Banaszeski de resolução de conflito de rules não ajudaria. Talvez verificar o uso de de LLVM (Low Level Virtual Machines) ao invés de threads: https://llvm.org/

Implementação atual:

Códigos disponíveis no servidor GIT, feature “ghk_thread”:



Graph	Description	Date	
	feature/ghk_thread origin/feature/ghk_thread **Gerada versao na mao para teste hipotetico de melhor caso com**	1 Sep 2018 13:24	ghk.martini
	Versao gerada para comparacao entre os metodos	31 Aug 2018 13:40	ghk.martini
	Ajuste na versao de thread pooling apos testes com mais cargas nas acoes	31 Aug 2018 12:35	ghk.martini
	Adicionado geracao de sinaleiro na versao de Pthreads visto o problema quando foi aumentada a carga computacional na versa	31 Aug 2018 11:49	ghk.martini
	Adicionado carga computacional nas acoes para poder comparar os 3 metodos	31 Aug 2018 11:19	ghk.martini
	Finalizada a geracao com Pthreads	30 Aug 2018 2:36	ghk.martini
	Gerado o arquivo premisses.cpp para versão PThreads	29 Aug 2018 2:26	ghk.martini
	Finalizada a geracao com thread pooling	28 Aug 2018 2:15	ghk.martini
	Primeiros acertos para gerar versão com thread pool	27 Aug 2018 9:33	ghk.martini
	Adicionado opcao de compilacao de Namespace com thread pool e pthreads	25 Aug 2018 18:58	ghk.martini
	Criada versão na mão com thread pool	22 Aug 2018 2:50	ghk.martini
	Gerado versao na mão com uso de Pthreads	21 Aug 2018 9:35	ghk.martini
	Gerado shell script para compilar 3 variacoes de arquivo CPP. Criado 3 diretorios replicados para modificar os CPPs para versoes	20 Aug 2018 0:26	ghk.martini
	Adicionado scripts para simplificar a compilacao de tudo em sequencia, usando o projeto ElectronicGate como exemplo. Verificar	19 Aug 2018 21:04	ghk.martini
	Added files to git ignore	9 Aug 2018 18:35	ghk.martini

Obrigado!