

OneAPI: uma abordagem para a computação heterogênea centrada no desenvolvedor



Ricardo Menotti
menotti@ufscar.br

Tiago da Conceição Oliveira
tiago.conceicao@fieb.org.br

14 de setembro de 2022

XXIII Simpósio em Sistemas Computacionais de Alto Desempenho

Departamento de Computação
Centro de Ciências Exatas e de Tecnologia
Universidade Federal de São Carlos

SENAI
CIMATEC

Antes de começarmos...

- <https://devcloud.intel.com/oneapi/>
- <https://github.com/menotti/sycl-wscad-2022/> Baixe este PDF
- <https://discord.gg/fdKYwvM> Acesse o Discord #oneapi-fpga-workshops
- <https://www.menti.com/> Informe o código!

Roteiro

1. Introdução
 - Objetivos
2. Arquiteturas
 - Processadores *multicore*
 - GPUs integradas
 - GPUs discretas
 - FPGAs
3. SYCL
 - oneAPI
4. Quiz
 - Teste seus conhecimentos
5. Background
 - Filtro Sobel
 - Transformada de Hough
 - Bitonic Sort
 - Sequências bitônicas
 - Criando sequências bitônicas
6. Práticas
 - Laboratórios
 - Para casa...

Introdução

Objetivos

- Aprender o básico para escrever programas SYCL
- Compreender o fluxo de desenvolvimento para CPUs, GPUs e FPGAs usando Intel® oneAPI
- Compreender os métodos de otimização mais comuns para CPUs, GPUs e FPGAs

Arquiteturas

Processadores multicore

Tabela 1: Aceleração de um programa que multiplica duas matrizes 4096 por 4096 (Leiserson et al., 2020).

Ver.	Implementação	Tempo (s)	GFLOPS	Abs.	Rel.	Pico (%)
1	Python	25.552,48	0,005	1	–	0,00
2	Java	2.372,68	0,058	11	10,8	0,01
3	C	542,67	0,253	47	4,4	0,03
4	Parallel loops	69,80	1,969	366	7,8	0,24
5	Parallel divide and conquer	3,80	36,180	6.727	18,4	4,33
6	plus vectorization	1,10	124,914	23.224	3,5	14,96
7	plus AVX intrinsics	0,41	337,812	62.806	2,7	40,45

Processadores multicore

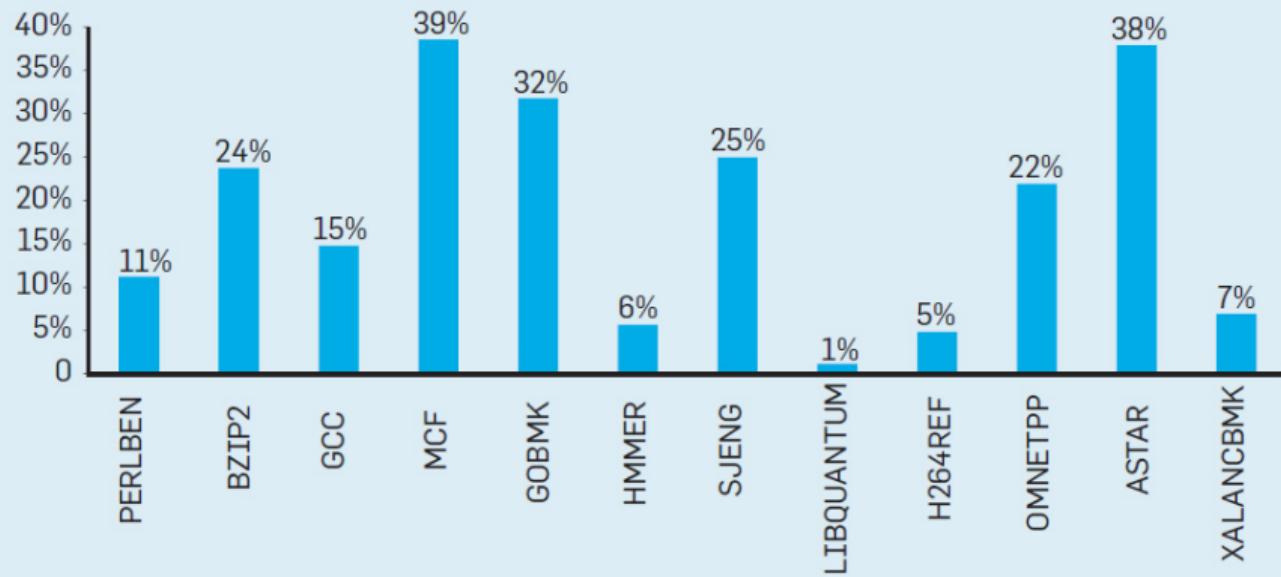


Figura 1: Percentual de instruções desperdiçadas em um processador Intel Core i7 para uma variedade de benchmarks de inteiros SPEC (Hennessy & Patterson, 2019).

GPUs integradas

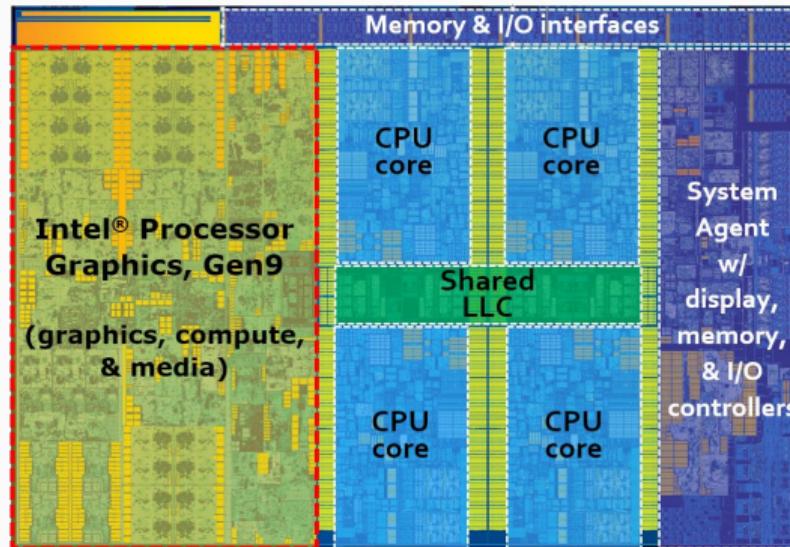


Figura 2: Processador Intel® Core™ i7 6700K de 4 núcleos com GPU Gen9 GT2 integrada. Nesta configuração a GPU tem 24 *slices* agrupados em 8 *subslashes*, cada um com 8 unidades de execução (EU).

GPUs integradas

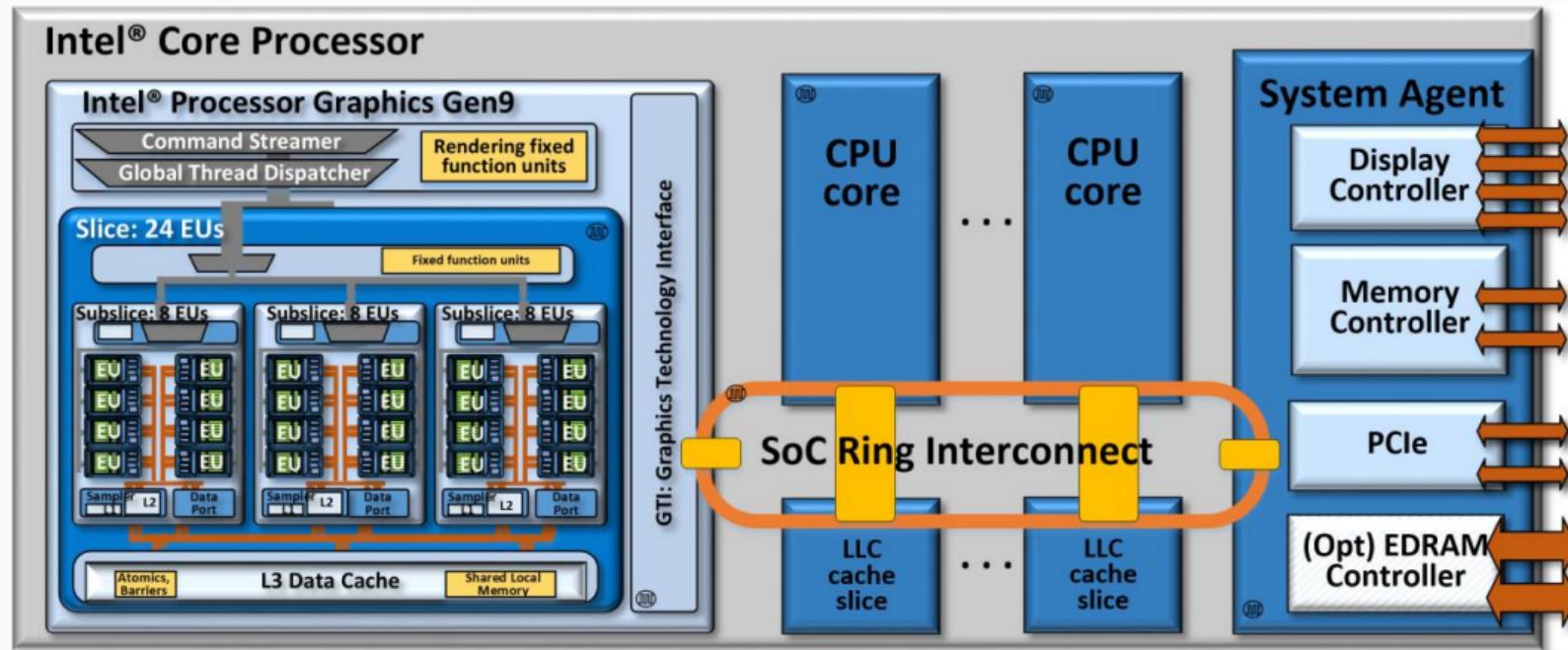


Figura 3: Interconexão em anel com múltiplos domínios de relógio.

GPUs integradas

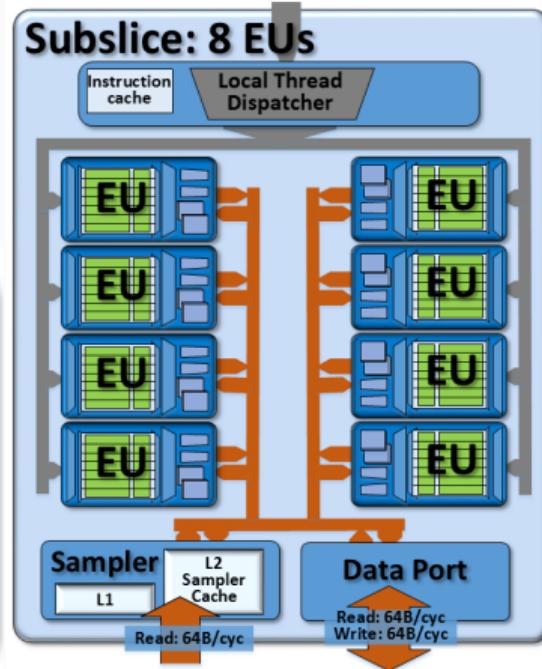
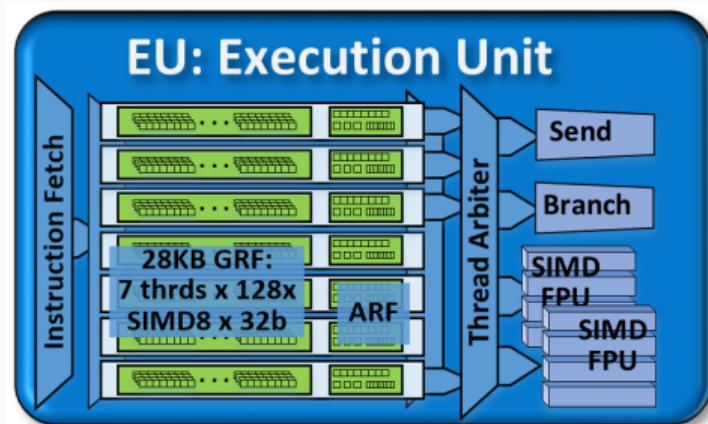
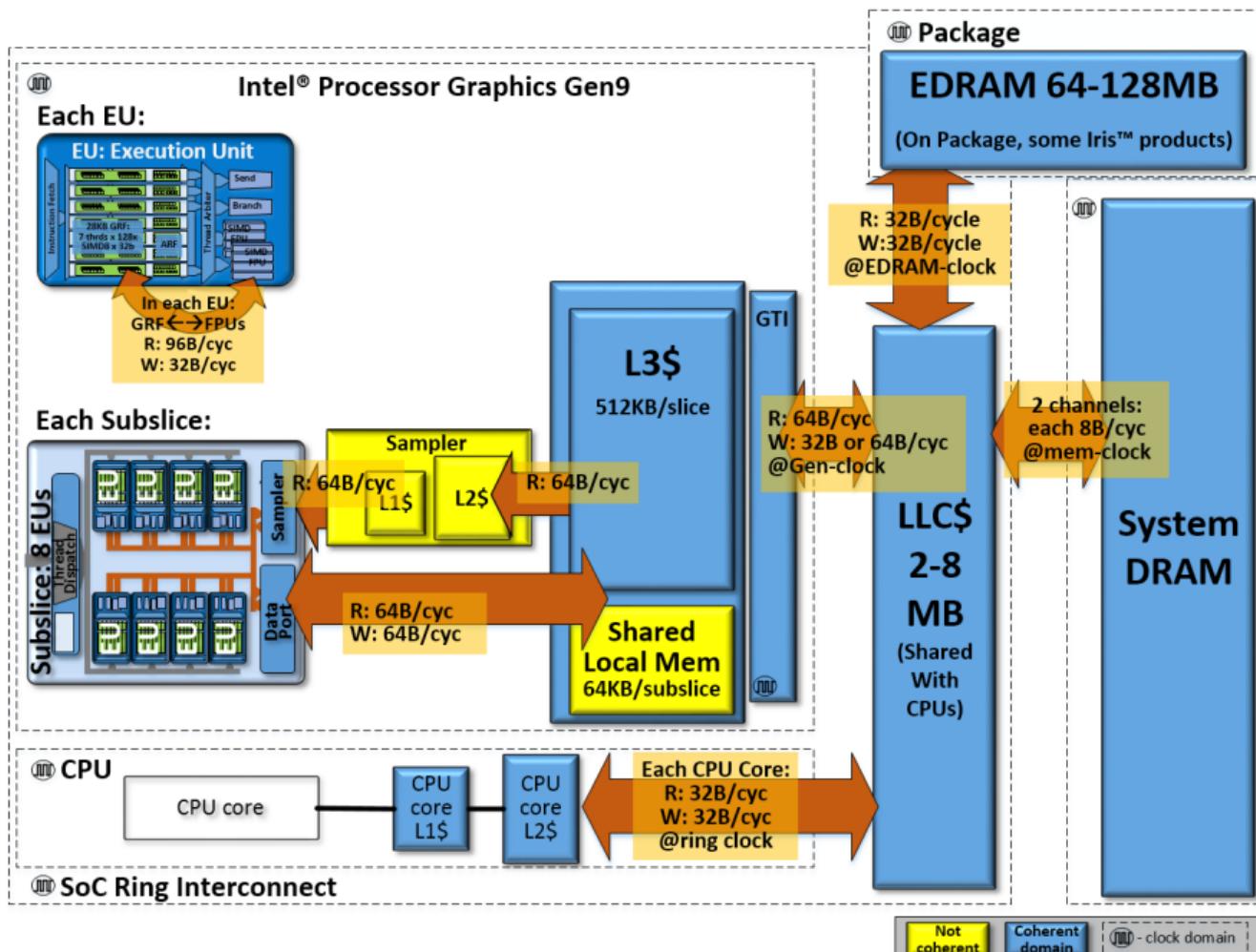


Figura 4: Um unidade de execução (EU) e seu agrupamento em um *subslice*.



GPUs discretas

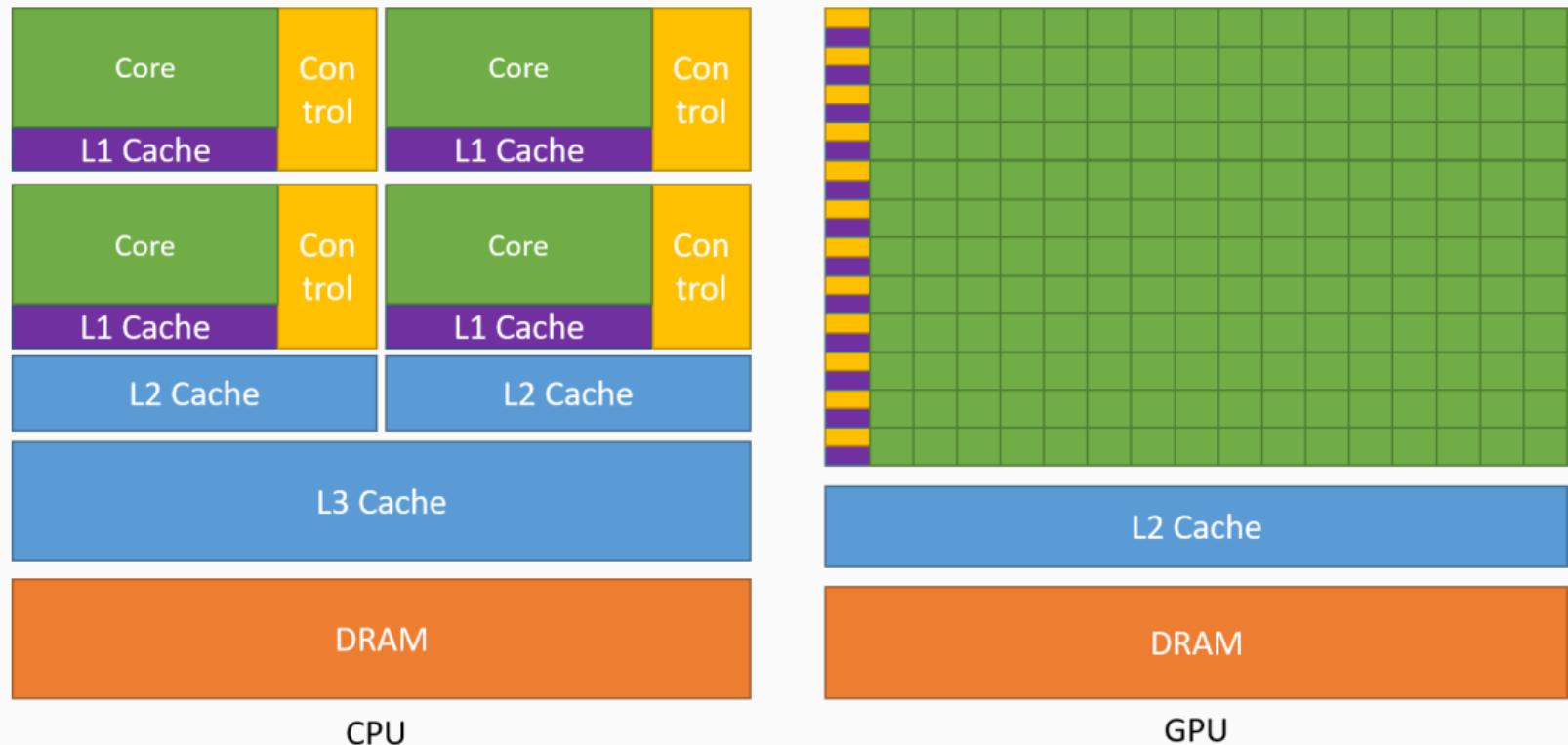


Figura 6: A GPU dedica mais transistores ao processamento de dados.

Computação Reconfigurável

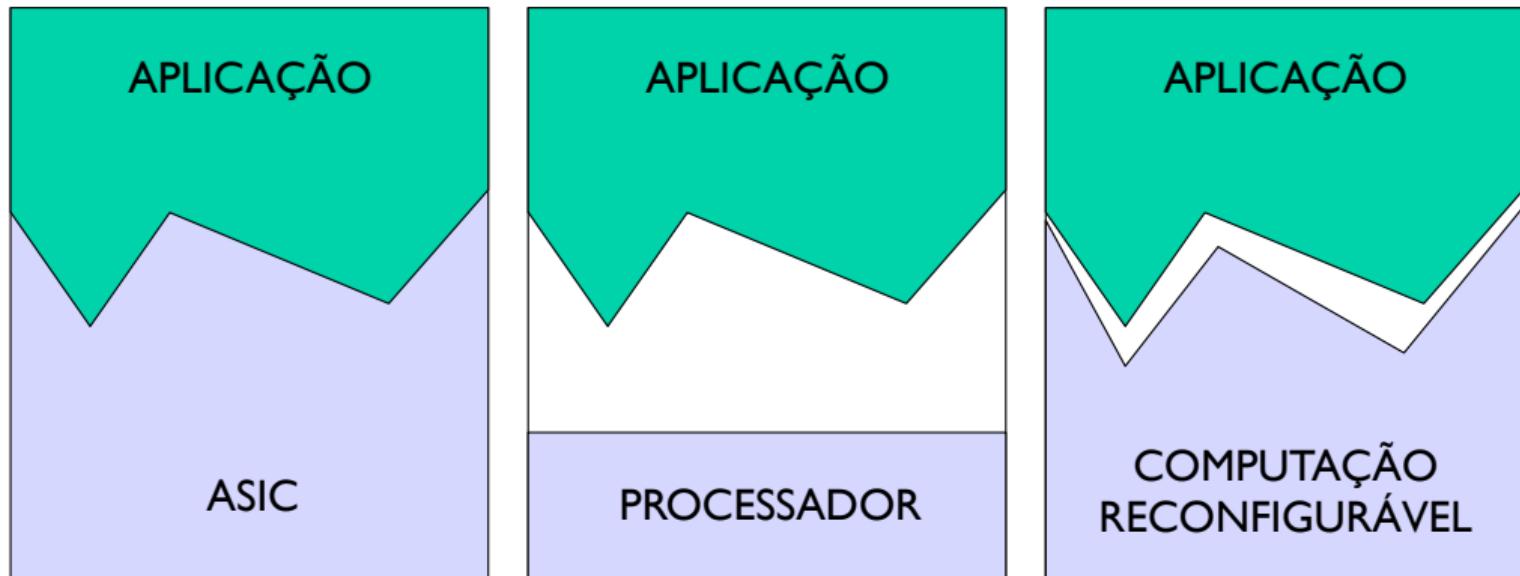
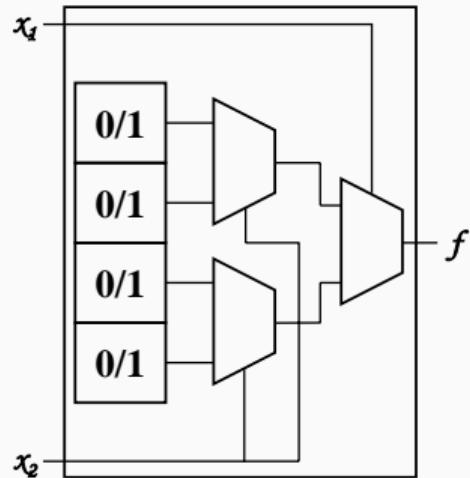


Figura 7: Computação reconfigurável comparada às soluções de *hardware dedicado* (ASIC) e *software* executando em processador de propósito geral (Menotti, 2010)

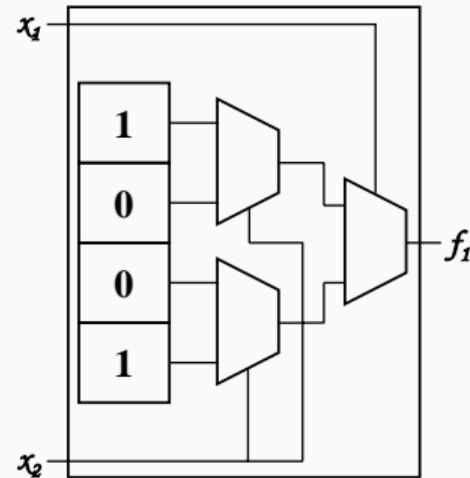
Look-Up Tables (LUTs)



(a)

x_1	x_2	f_1
0	0	1
0	1	0
1	0	0
1	1	1

(b)



(c)

Figura 8: Implementação de uma função lógica em uma LUT (Menotti, 2010)

Programmable Acceleration Card with Arria® 10 GX FPGA (\$3.9K)



Programmable Acceleration Card with Stratix® 10 SX FPGA (\$7.9K)



Fluxo de desenvolvimento para FPGAs

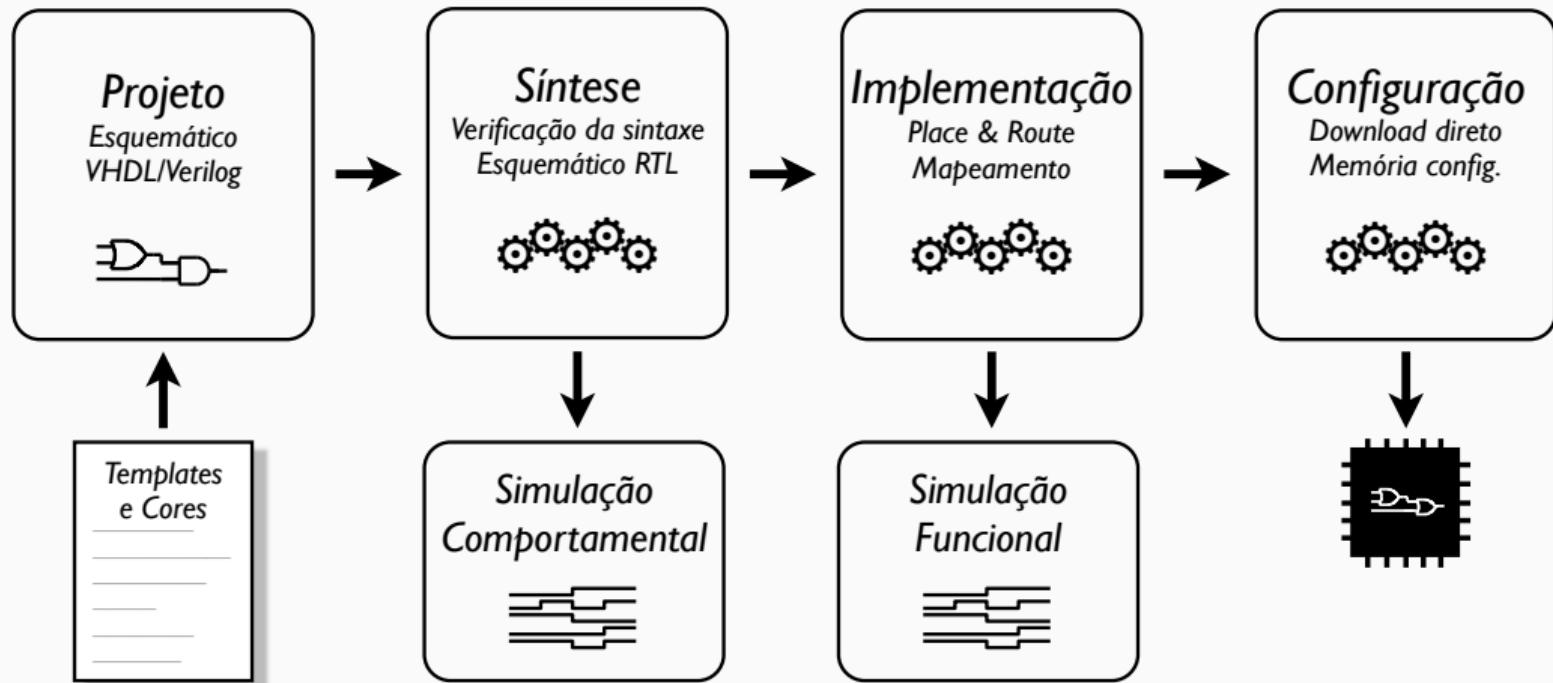


Figura 9: Fluxo de desenvolvimento para FPGAs (Menotti, 2010)

SYCL

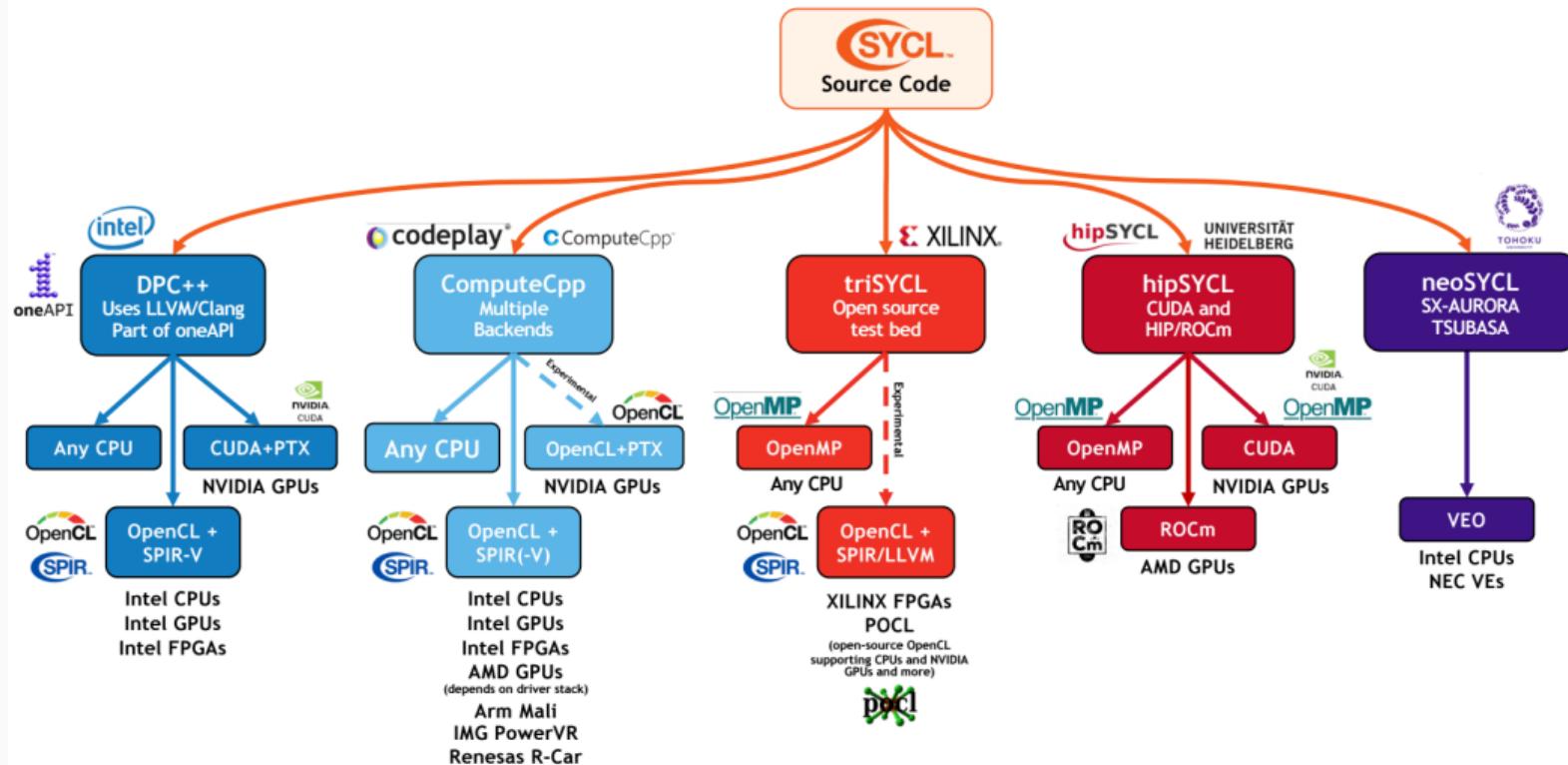


Figura 10: Algumas das implementações SYCL disponíveis atualmente

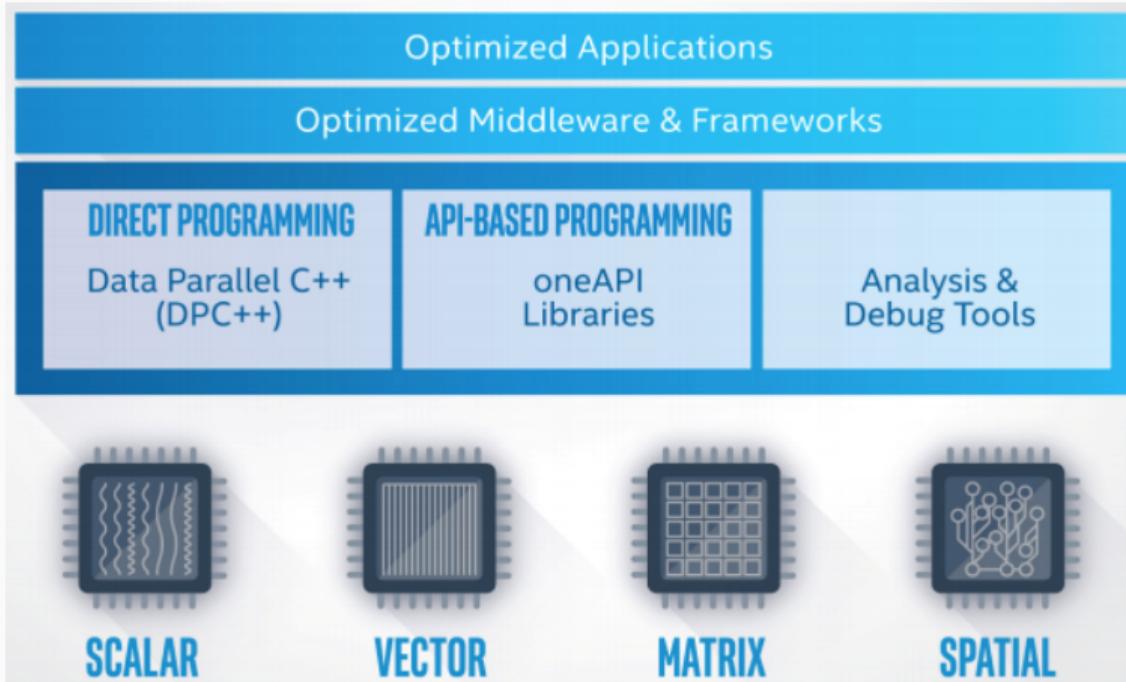


Figura 11: Uma abordagem para a computação heterogênea centrada no desenvolvedor

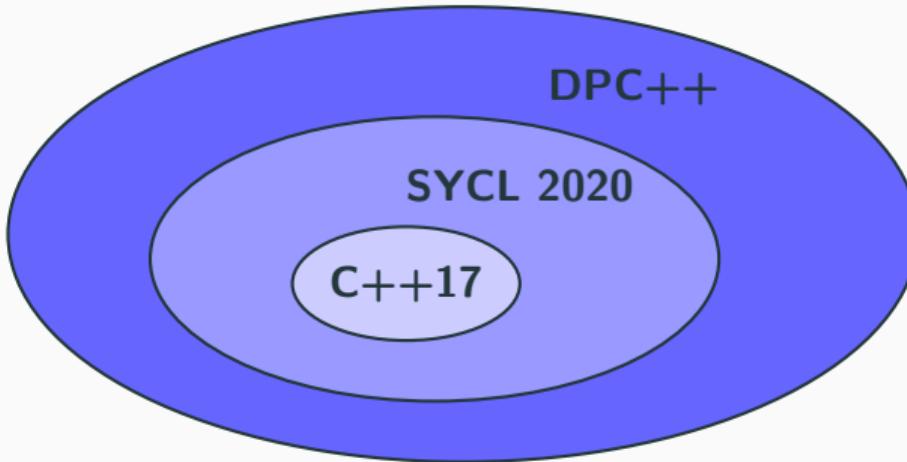


Figura 12: Relação de DPC++ com SYCL e C++ (Intel Corp., 2020)

Templates

```
1 #include <iostream>
2
3 template <class T>
4 T GetMax (T a, T b) {
5     T result;
6     result = (a > b) ? a : b;
7     return (result);
8 }
9
10 int main () {
11     int i = 7, j = 5, k;
12     float l = 1.0, m = 5.5, n;
13     k = GetMax<int>(i, j);
14     n = GetMax<float>(l, m);
15     std::cout << k << std::endl;
16     std::cout << n << std::endl;
17     return 0;
18 }
```

São funções/classes especiais que podem operar com tipos genéricos, cuja funcionalidade pode ser adaptada a mais de um tipo ou classe sem repetir o código inteiro para cada tipo.

Saída:

7

5.5

Experimente este exemplo mais complexo depois...

Lambdas

```
1 #include <iostream>
2 #include <typeinfo>
3
4 int main()
5 {
6     int i = 1;
7     int j = 2;
8
9     auto f = [&i, j](int a) -> double
10    { return i + j + a; };
11
12    i = 4;
13    j = 8;
14
15    std::cout << f(1)  << std::endl;
16    std::cout << typeid(f(42)).name()
17    << std::endl;
```

São uma maneira conveniente de definir um objeto de função anônimo.

Saída:

7
d

Modifique este exemplo depois...

Inferência de tipos (auto)¹

```
1 std::vector<int> v {2, 7, 42};  
2 for (std::vector<int>::iterator e = v.begin(); e != v.end(); ++e)  
3     std::cout << *e << std::endl;
```

¹C++ for Python Programmers

Inferência de tipos (auto)¹

```
1 std::vector<int> v {2, 7, 42};  
2 for (std::vector<int>::iterator e = v.begin(); e != v.end(); ++e)  
3     std::cout << *e << std::endl;
```

```
1 std::vector v {2, 7, 42};          // Polêmica: C++ > Python  
2 for (auto e : v)  
3     std::cout << e << std::endl;
```

¹C++ for Python Programmers

Inferência de tipos (auto)¹

```
1 std::vector<int> v {2, 7, 42};  
2 for (std::vector<int>::iterator e = v.begin(); e != v.end(); ++e)  
3     std::cout << *e << std::endl;
```

```
1 std::vector v {2, 7, 42};          // Polêmica: C++ > Python  
2 for (auto e : v)  
3     std::cout << e << std::endl;
```

```
1 // Desafio: tente especificar o tipo da variável acc  
2 auto acc = buf.get_access<sycl::access::mode::read_write>(cgh);
```

¹C++ for Python Programmers

Quiz

Acesse para testar seus conhecimentos!

www.kahoot.it (melhor com celular)



Background

Filtro Sobel

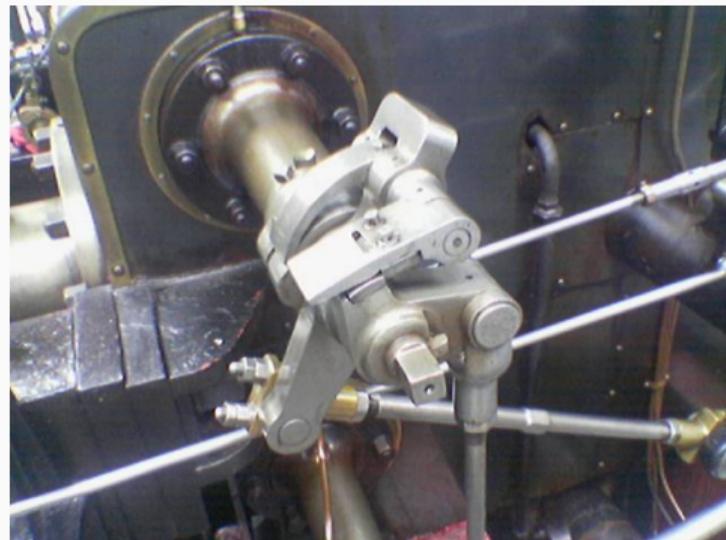


Figura 13: Exemplo da aplicação do filtro Sobel

Filtro Sobel

```
1   float sobel_x[3][3] =  
2   { { -1, 0, 1 },  
3     { -2, 0, 2 },  
4     { -1, 0, 1 } };  
5  
6   float sobel_y[3][3] =  
7   { { -1, -2, -1 },  
8     { 0, 0, 0 },  
9     { 1, 2, 1 } };  
10  for (int x=1; x < width-1; x++)  
11    for (int y=1; y < height-1; y++) {  
12      pixel_x = (sobel_x[0][0] * bufg[width * (y-1) + (x-1)][0])  
13        + (sobel_x[0][1] * bufg[width * (y-1) + x ][0])  
14        + (sobel_x[0][2] * bufg[width * (y-1) + (x+1)][0])  
15        + (sobel_x[1][0] * bufg[width * y + (x-1)][0])  
16        + (sobel_x[1][1] * bufg[width * y + x ][0])  
17        + (sobel_x[1][2] * bufg[width * y + (x+1)][0])  
18        + (sobel_x[2][0] * bufg[width * (y+1) + (x-1)][0])  
19        + (sobel_x[2][1] * bufg[width * (y+1) + x ][0])  
20        + (sobel_x[2][2] * bufg[width * (y+1) + (x+1)][0]);  
21      pixel_y = (sobel_y[0][0] * bufg[width * (y-1) + (x-1)][0])  
22        + (sobel_y[0][1] * bufg[width * (y-1) + x ][0])  
23        + (sobel_y[0][2] * bufg[width * (y-1) + (x+1)][0])  
24        + (sobel_y[1][0] * bufg[width * y + (x-1)][0])  
25        + (sobel_y[1][1] * bufg[width * y + x ][0])  
26        + (sobel_y[1][2] * bufg[width * y + (x+1)][0])  
27        + (sobel_y[2][0] * bufg[width * (y+1) + (x-1)][0])  
28        + (sobel_y[2][1] * bufg[width * (y+1) + x ][0])  
29        + (sobel_y[2][2] * bufg[width * (y+1) + (x+1)][0]);  
30      int val = (int)sqrt((pixel_x * pixel_x) + (pixel_y * pixel_y));  
31      if(val < 0) val = 0;  
32      if(val > 255) val = 255;  
33      out[height * y + x][2] = (unsigned char)val;  
34      out[height * y + x][1] = (unsigned char)val;  
35      out[height * y + x][0] = (unsigned char)val;  
36    }
```

Transformada de Hough

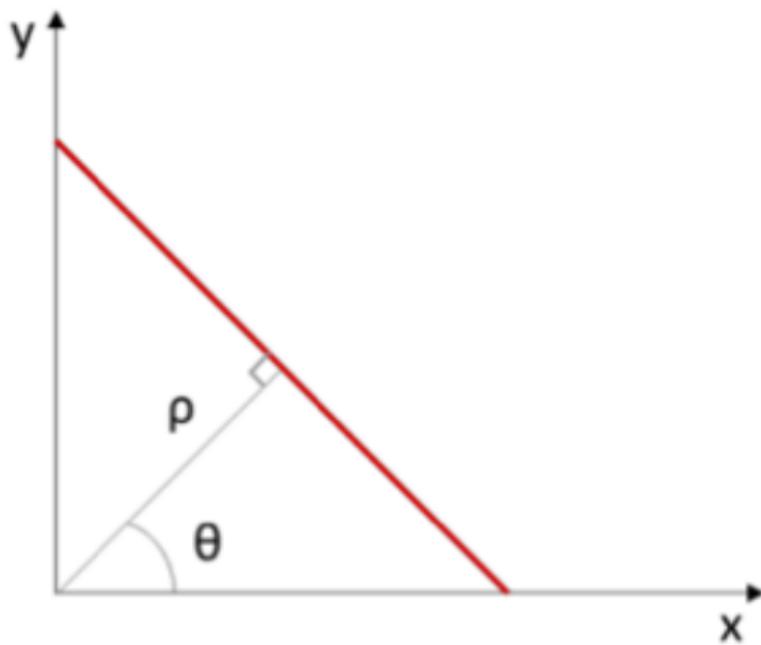


Figura 14: Representação dos valores de ρ e θ na equação

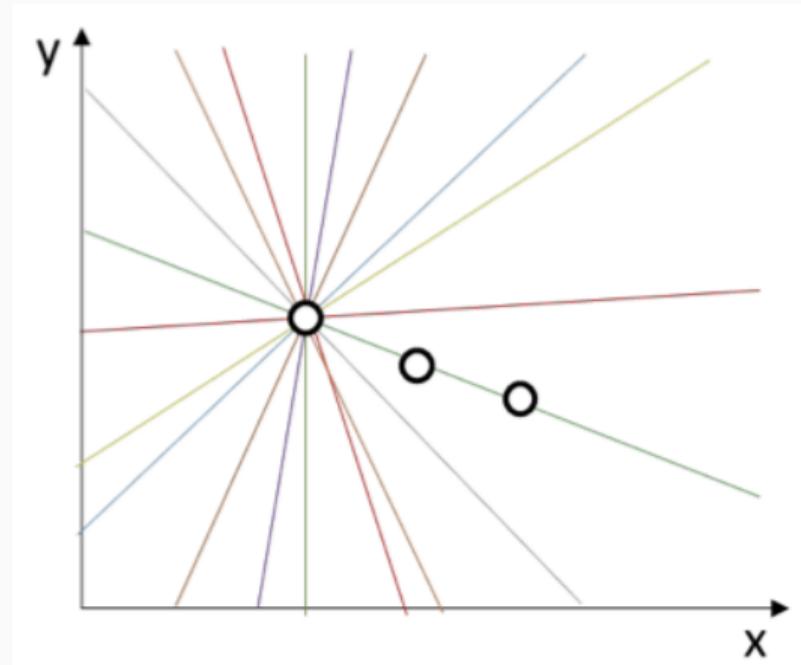


Figura 15: Várias linhas que interceptam um ponto

Transformada de Hough

```
1 for (uint y=0; y<HEIGHT; y++) {  
2     for (uint x=0; x<WIDTH; x++) {  
3         unsigned short int increment = 0;  
4         if (_pixels[(WIDTH*y)+x] != 0) {  
5             increment = 1;  
6         } else {  
7             increment = 0;  
8         }  
9         for (int theta=0; theta<THETAS; theta++) {  
10             int rho = x*_cos_table[theta] + y*_sin_table[theta];  
11             _accumulators[(THETAS*(rho+RHOS))+theta] += increment;  
12         }  
13     }  
14 }
```

Bitonic Sort

- Algoritmo de divisão e conquista;
- Baseado no Merge Sort;
- Utiliza sequências bitônicas para realizar a ordenação.

Sequências bitônicas

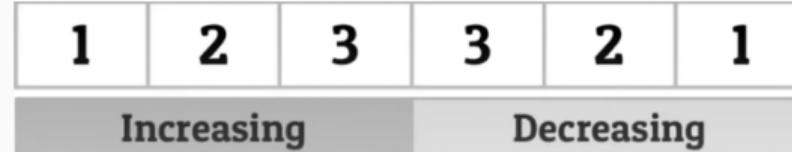


Figura 16: Exemplo de sequência bitônica.



Figura 17: Dividindo uma lista de valores em sequências bitônicas de 2 números cada.

Criando sequências bitônicas

1	3	5	7	8	6	4	2
Bitonic Sequence				Bitonic Sequence			

Figura 18: Agrupando duas sequências em uma.

1	2	3	4	5	6	7	8
Bitonic Sequence							

Figura 19: Sequência resultante do processo de BitonicSort.

Práticas

Vamos para a prática!?

<http://devcloud.intel.com/oneapi/>

Mãos à obra!

1. Acesse o Jupyter!
2. Abra um terminal lá
3. Faça um clone do repositório:
 - `git clone https://github.com/menotti/sycl-wscad-2022`

Para casa...

- **Atenção: por favor, não façam durante o curso!**
- Experimente o Notebook lab3/Bonus_FPGA.ipynb

Referências

-  Hennessy, J. L. & Patterson, D. A. (2019). A New Golden Age for Computer Architecture. *Commun. ACM*, 62(2), 48–60. <https://doi.org/10.1145/3282307>
-  Intel Corp. (2020). *FPGA Add-On for oneAPI Base Toolkit(Beta)*. <https://software.intel.com/content/www/us/en/develop/tools/oneapi/components/fpga.html>
-  Leiserson, C. E., Thompson, N. C., Emer, J. S., Kuszmaul, B. C., Lampson, B. W., Sanchez, D. & Schardl, T. B. (2020). There's plenty of room at the Top: What will drive computer performance after Moore's law? *Science*, 368(6495), eaam9744. <https://doi.org/10.1126/science.aam9744>
-  Menotti, R. (2010). *LALP: uma linguagem para exploração do paralelismo de loops em computação reconfigurável* (tese de dout.). Universidade de São Paulo. <https://doi.org/10.11606/T.55.2010.tde-17082010-151100>

Obrigado!

OneAPI: uma abordagem para a computação heterogênea centrada no desenvolvedor



Ricardo Menotti
menotti@ufscar.br

Tiago da Conceição Oliveira
tiago.conceicao@fieb.org.br

14 de setembro de 2022

XXIII Simpósio em Sistemas Computacionais de Alto Desempenho

Departamento de Computação
Centro de Ciências Exatas e de Tecnologia
Universidade Federal de São Carlos

SENAI
CIMATEC