# Создание приложений для суперкомпьютеров

2022

# План


- Модели параллельных вычислений.
- Методы извлечения параллелизма.
- Параллельные алгоритмы.

*Суть программирования — в управлении сложностью.*

Б. Керниган

- ассемблерные вставки = сложность
- ручная векторизация кода = сложность
- функция `main` на 10000 строк = сложность
- язык Си = сложность
- общее состояние = сложность

*Суть программирования — в управлении сложностью.*

Б. Керниган

- ассемблерные вставки = сложность
- ручная векторизация кода = сложность
- функция `main` на 10000 строк = сложность
- язык Си = сложность
- общее состояние = сложность

# Как управлять общим состоянием?

Веб-приложения:

храним состояние на клиенте, передаем на сервер по необходимости.

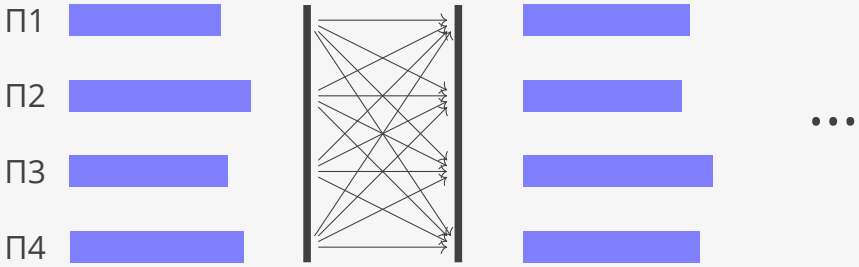
Анализ данных:

копируем состояние на каждом шаге обработки.

Суперкомпьютинг:

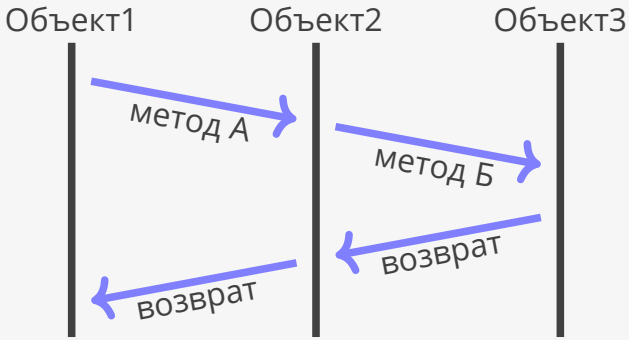
изменяем состояние вручную.

# Модель Bulk Synchronous Parallel (BSP)




Leslie G. Valiant “A bridging model for parallel computation.” Communications of the ACM 33.8 (1990): 103–111.

# Модель акторов



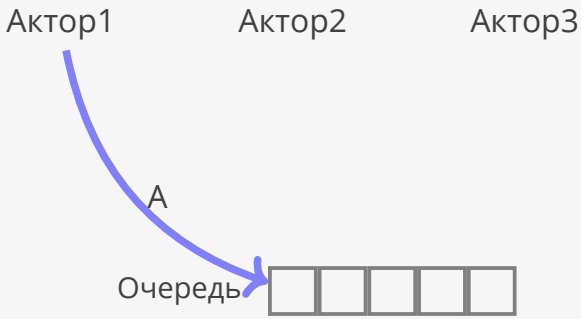
# Модель акторов

Актор1 Актор2 Актор3

Очередь 

C. Hewitt, P. Bishop, R. Steiger “A universal modular ACTOR formalism for artificial intelligence.” Proceedings of IJCAI (1973): 235–245.

# Модель акторов



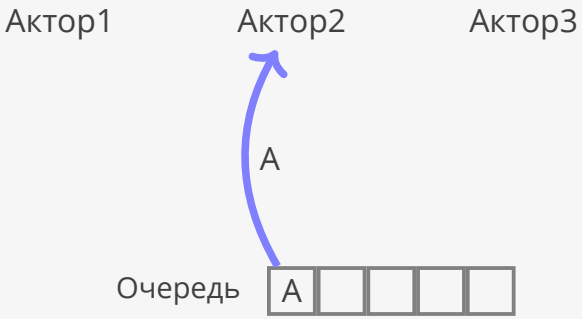

C. Hewitt, P. Bishop, R. Steiger “A universal modular ACTOR formalism for artificial intelligence.” Proceedings of IJCAI (1973): 235–245.

# Модель акторов

Актор1 Актор2 Актор3


C. Hewitt, P. Bishop, R. Steiger “A universal modular ACTOR formalism for artificial intelligence.” Proceedings of IJCAI (1973): 235–245.

# Модель акторов




C. Hewitt, P. Bishop, R. Steiger “A universal modular ACTOR formalism for artificial intelligence.” Proceedings of IJCAI (1973): 235–245.

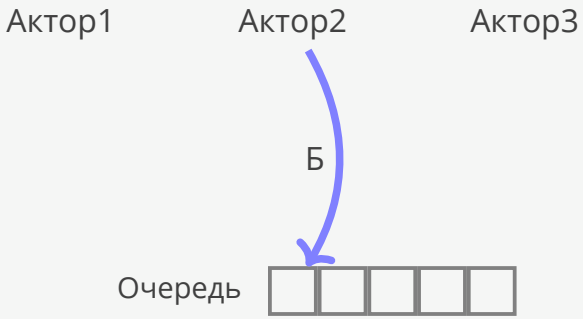
# Модель акторов

Актор1 Актор2 Актор3

Очередь 

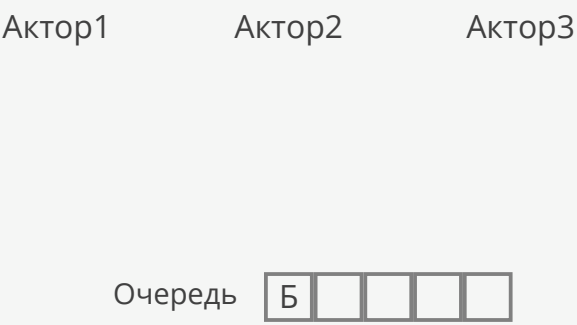
C. Hewitt, P. Bishop, R. Steiger “A universal modular ACTOR formalism for artificial intelligence.” Proceedings of IJCAI (1973): 235–245.

# Модель акторов



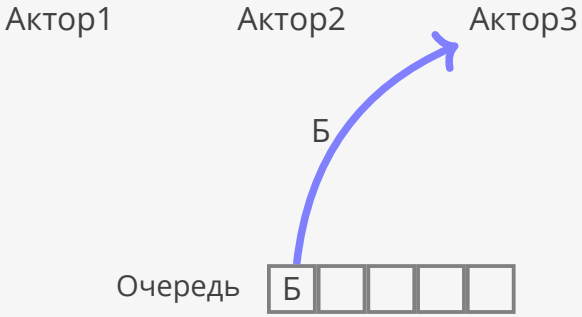

C. Hewitt, P. Bishop, R. Steiger “A universal modular ACTOR formalism for artificial intelligence.” Proceedings of IJCAI (1973): 235–245.

# Модель акторов




C. Hewitt, P. Bishop, R. Steiger “A universal modular ACTOR formalism for artificial intelligence.” Proceedings of IJCAI (1973): 235–245.

# Модель акторов




C. Hewitt, P. Bishop, R. Steiger “A universal modular ACTOR formalism for artificial intelligence.” Proceedings of IJCAI (1973): 235–245.

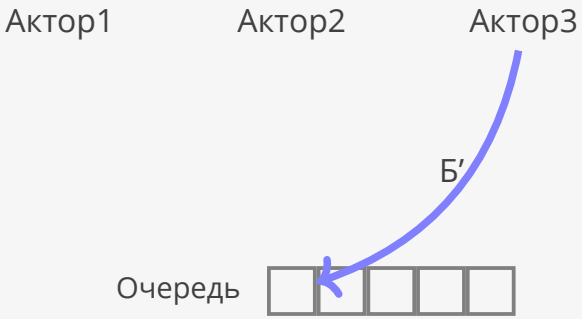
# Модель акторов

Актор1 Актор2 Актор3

Очередь 

C. Hewitt, P. Bishop, R. Steiger “A universal modular ACTOR formalism for artificial intelligence.” Proceedings of IJCAI (1973): 235–245.

# Модель акторов



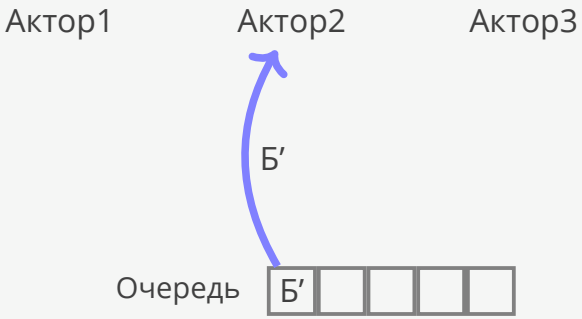

C. Hewitt, P. Bishop, R. Steiger “A universal modular ACTOR formalism for artificial intelligence.” Proceedings of IJCAI (1973): 235–245.

# Модель акторов

Актор1 Актор2 Актор3


C. Hewitt, P. Bishop, R. Steiger “A universal modular ACTOR formalism for artificial intelligence.” Proceedings of IJCAI (1973): 235–245.

# Модель акторов




C. Hewitt, P. Bishop, R. Steiger “A universal modular ACTOR formalism for artificial intelligence.” Proceedings of IJCAI (1973): 235–245.

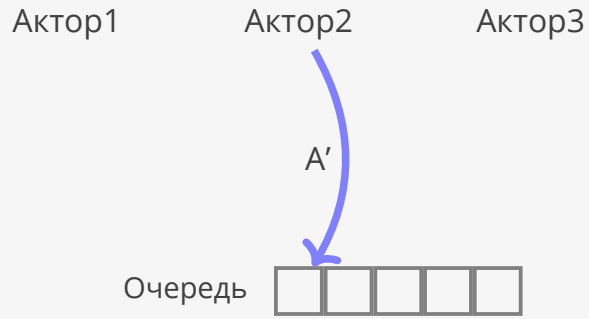
# Модель акторов

Актор1 Актор2 Актор3

Очередь 

C. Hewitt, P. Bishop, R. Steiger “A universal modular ACTOR formalism for artificial intelligence.” Proceedings of IJCAI (1973): 235–245.

# Модель акторов




C. Hewitt, P. Bishop, R. Steiger “A universal modular ACTOR formalism for artificial intelligence.” Proceedings of IJCAI (1973): 235–245.

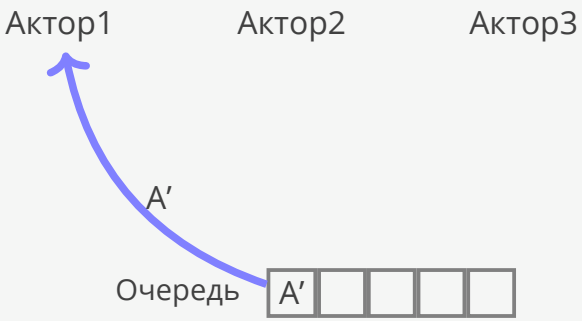
# Модель акторов

Актор1 Актор2 Актор3

Очередь 


C. Hewitt, P. Bishop, R. Steiger “A universal modular ACTOR formalism for artificial intelligence.” Proceedings of IJCAI (1973): 235–245.

# Модель акторов




C. Hewitt, P. Bishop, R. Steiger “A universal modular ACTOR formalism for artificial intelligence.” Proceedings of IJCAI (1973): 235–245.

# Модель акторов

Актор1 Актор2 Актор3

Очередь 

C. Hewitt, P. Bishop, R. Steiger “A universal modular ACTOR formalism for artificial intelligence.” Proceedings of IJCAI (1973): 235–245.

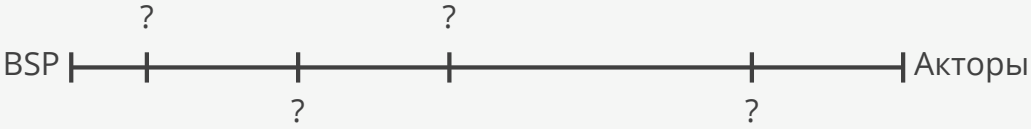
# Преимущества и недостатки

Массовый параллелизм (BSP)

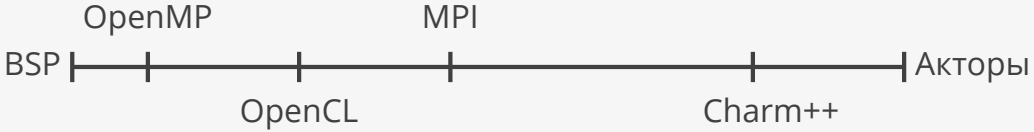
- Легко реализовать контрольные точки восстановления.
- Легко программировать.
- Глобальная синхронизация может быть медленной и ненужной.

Акторы

- Нет глобальной синхронизации.
- Очередь обрабатывается параллельно.
- Сложно программировать.
- Идеальна для задач со сложными информационными зависимостями.




MPI, OpenMP, OpenCL, Charm++


OpenMP
MPI
BSP
Акторы
OpenCL
Charm++

# Параллелизм по данным

Последовательный код:

```
float interval = 2.0f;
std::valarray<float> arr(1000);

for (size_t i=0; i<arr.size(); ++i) {
    arr[i] = bisection(-interval, interval,
        Equation(...), eps);
}
```

Параллельный код:

```
float interval = 2.0f;
std::valarray<float> arr(1000);
#pragma omp parallel
{
    #pragma omp for
    for (size_t i=0; i<arr.size(); ++i) {
        arr[i] = bisection(-interval, interval,
            Equation(...), eps);
    }
}
```

Динамическая балансировка нагрузки:

```
float interval = 2.0f;
std::valarray<float> arr(1000);
#pragma omp parallel
{
    #pragma omp for schedule(dynamic,1)
    for (size_t i=0; i<arr.size(); ++i) {
        arr[i] = bisection(-interval, interval,
            Equation(...), eps);
    }
}
```

# Векторизация

$$p(x,y) = -g\zeta(x,y) - \frac{1}{2}\left(\phi_x^2 + \phi_y^2 + \phi_z^2\right)$$

$p$ давление
$\zeta$ подъем поверхности
$\phi$ потенциал скорости

Массив структур `phi`:

```
const float g = 9.8f;
for (int i=0; i<nx; ++i) {
    for (int j=0; j<ny; ++j) {
        int idx = i*nx + j;
        p[idx] = -g*zeta[idx] - 0.5f*(pow2(phi[idx].x)
            + pow2(phi[idx].y) + pow2(phi[idx].z));
    }
}
```

Структура массивов phi_x, phi_y, phi_z:

```
for (int i=0; i<nx; ++i) {
    for (int j=0; j<ny; ++j) {
        int idx = i*nx + j;
        phi_x[idx] = phi[idx].x;
        phi_y[idx] = phi[idx].y;
        phi_z[idx] = phi[idx].z;
    }
}
```

Новый цикл:

```
int n = nx*ny;
for (int i=0; i<n; ++i) {
    p[i] = -g*zeta[i] - 0.5f*(pow2(phi_x[i])
                            + pow2(phi_y[i])
                            + pow2(phi_z[i]));
}
```

Новый цикл:

```
int n = nx*ny;
for (int i=0; i<n; ++i) {
    p[i] = -g*zeta[i] - 0.5f*(pow2(phi_x[i])
                            + pow2(phi_y[i])
                            + pow2(phi_z[i]));
}
```

Если p, zeta, phi_x, phi_y, phi_z имеют тип std::valarray<float>:

```
p = -g*zeta - 0.5f*(pow2(phi_x) + pow2(phi_y) + pow2(phi_z));
```

Исходная формула: $p(x,y) = -g\zeta(x,y) - \frac{1}{2}\left(\phi_x^2 + \phi_y^2 + \phi_z^2\right)$.

# Выводы

- Для 80% случаев достаточно `#pragma omp parallel for`.
- Библиотеки многомерных массивов ускоряют программу за счет векторизации.
- Итераторы и большинство алгоритмов STL бесполезны для многомерных массивов.

# Вихрь Мерсенна

## Последовательная версия

```
unsigned long seed = ...;
sgenrand(seed);          // инициализация генератора
double d = genrand();    // генерация псевдо-случайного числа
```

## Параллельная версия

Программа для динамического создания генераторов (DCMT):

```
init_dc();               // инициализация генератора
mtconf = genmt(period); // генерация Вихря Мерсенна
```

Основная программа:

```
unsigned long seed = ...;
sgenrand(seed, mtconf); // из программы 1
double d = genrand();    // генерация псевдо-случайного числа
```

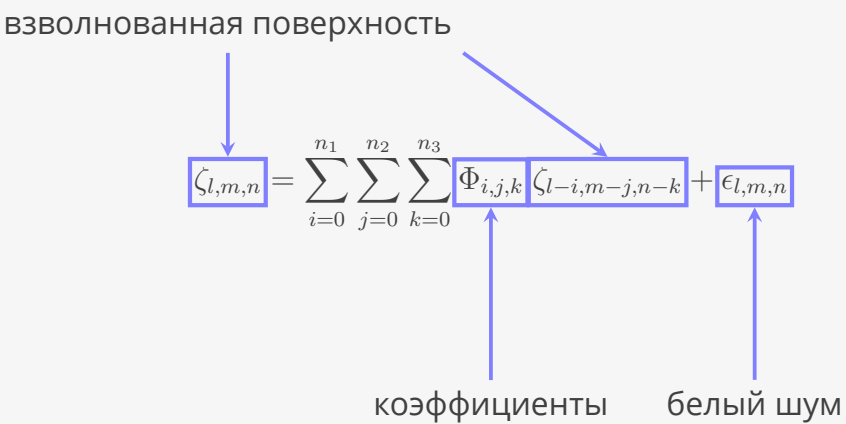
# Особенности

- Программа DCMT последовательная, основная — параллельная.
- У параллельных генераторов меньший период.
- Последовательности чисел некоррелированны друг с другом.
- Параллелизм по данным.

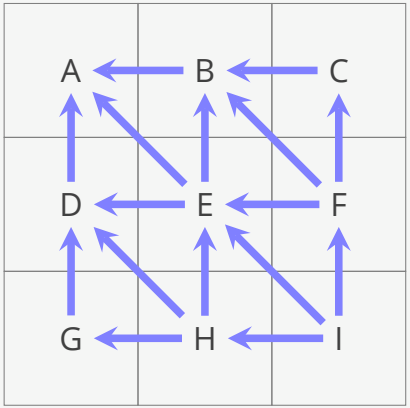
# Параллельная модель авторегрессии

$$\zeta_{l,m,n} = \sum_{i=0}^{n_1} \sum_{j=0}^{n_2} \sum_{k=0}^{n_3} \Phi_{i,j,k} \; \zeta_{l-i,m-j,n-k} + \epsilon_{l,m,n}$$

# Параллельная модель авторегрессии



# Информационные зависимости



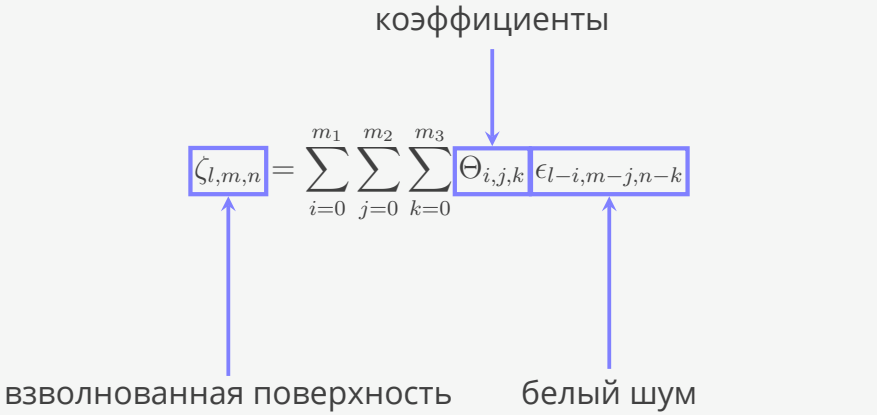

Двухмерная модель

- Размер блоков ограничен сверху и снизу.
- Конвейер + параллелизм по данным.

# Параллельная модель скользящего среднего

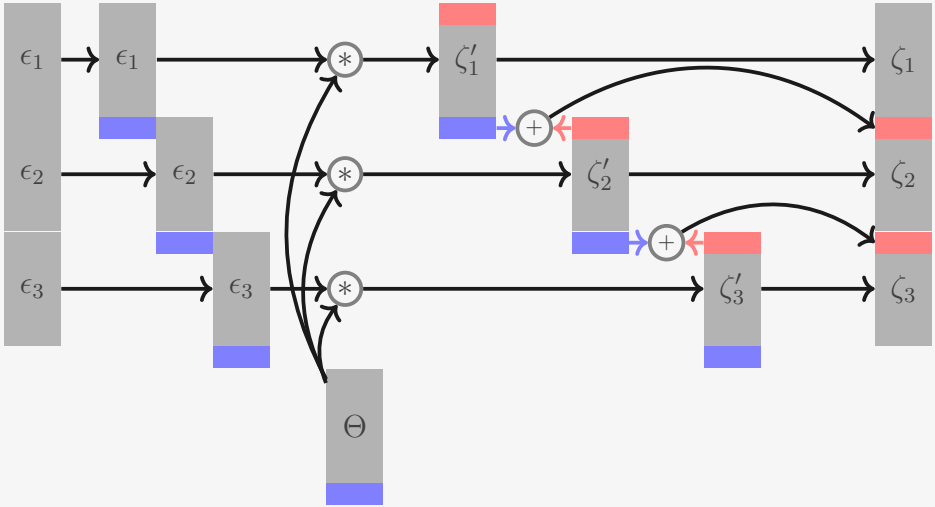
$$\zeta_{l,m,n} = \sum_{i=0}^{m_1} \sum_{j=0}^{m_2} \sum_{k=0}^{m_3} \Theta_{i,j,k} \ \epsilon_{l-i,m-j,n-k}$$

# Параллельная модель скользящего среднего



Перепишем модель в виде свертки и воспользуемся одноименной теоремой:

$$\begin{aligned}\zeta_{l,m,n} &= \sum_{i=0}^{m_1}\sum_{j=0}^{m_2}\sum_{k=0}^{m_3}\Theta_{i,j,k}\ \epsilon_{l-i,m-j,n-k} \\ &= \Theta * \epsilon \\ &= \mathcal{F}^{-1}\{\mathcal{F}\{\Theta\}\,\mathcal{F}\{\epsilon\}\}\end{aligned}$$




Karas Pavel, David Svoboda.
"Algorithms for efficient computation of convolution."
Design and Architectures for Digital Signal Processing. InTech, 2013.

# Особенности

- Очень много уровней параллелизма.
- Размер блоков ограничен сверху и снизу.
- Конвейер + параллелизм по данным + векторизация.
- Идеален для видеокарты.

# «Алгоритм» ускорения программы

Пока программа не достаточно ускорена

- найти часть, занимающую большую часть времени (hot spot),
- найти параллелизм по операциям и переписать через конвейер,
- найти параллелизм по данным и переписать, используя потоки,
- векторизовать оставшиеся куски кода.

# Ссылки


- R. Fielding Architectural Styles and the Design of Network-based Software Architectures, PhD, 2000.
- Makoto Matsumoto, Takuji Nishimura. Mersenne Twister: a 623-dimensionally equidistributed uniform pseudo-random number generator, ACM Transactions on Modeling and Computer Simulation 8.1 (1998): 3-30.
- Makoto Matsumoto, Takuji Nishimura. Dynamic creation of pseudorandom number generators Monte Carlo and Quasi-Monte Carlo Methods 2000 (1998): 56-69.