Спецкурс «Параллельная обработка больших графов»

Лекция 7

А.С. Семенов

dislab.org

Проблемы и подходы к решению задач обработки графов в рамках одного вычислительного узла

Алгоритм U. Brandes (2001)

BC-Brandes (G) // G – связный неориентированный без весов
for all v∈V **do** BC[v] = 0 **end for**
for all s∈V **do**
    BrandesInitS (s)
    ShortestPathsCounting (s)
    DependencyAccumulation ()
end for

Алгоритм Brandes, инициализация

BrandesInitS (s)
initEmptyStack(S) // S - пустой стек
for all t∈V **do**
    initEmptyList(P[t]);
    $\sigma[t] = 0; d[t] = -1; \delta[v] = 0$
end for
$\sigma[s] = 1; d[s] = 0$

Подсчет числа кратчайших путей

ShortestPathsCounting (Q)
initQueue(Q, *s*) // Q – очередь
while Q <> {}
    v = deque(Q); push(S, *v*)
    for all w∈Adj[v]
        if $d[w] < 0$
            enqueue(Q, *w*)
            $d[w] = d[v] + 1$
        end if
        if $d[w] == d[v] + 1$
            $\sigma[w] = \sigma[w] + \sigma[v]$
            append(P[*w*], *v*)
        end if
    end for
end while

Суммирование

DependencyAccumulation ()

while S <> {}

w = pop(S)

for all $v \in P[w]$

#pragma omp atomic

$$\delta[v] = \delta[v] + \frac{\sigma[v]}{\sigma[w]} \cdot (1 + \delta[w])$$

if *w* <> *s*

BC[*w*] = BC[*w*] + $\delta[w]$

end if

end for

end while

Суммирование

DependencyAccumulation ()

phase = phase - 1

#pragma omp parallel for

for all *w*∈S[phase]

dsw = 0

sw = $\sigma[w]$

for all *v*∈Succ[*w*]

$$dsw = dsw + \frac{sw}{\sigma[v]} \cdot (1 + \delta[v])$$

end for

$dsw = \delta[w]$

BC[*w*] = BC[*w*] + *dsw*

end for

Как измерять

- Первая итерация – «прогрев» системы
- Использовать как минимум несколько итераций

Резюме: проблемы и подходы к решению задач в рамках одного узла

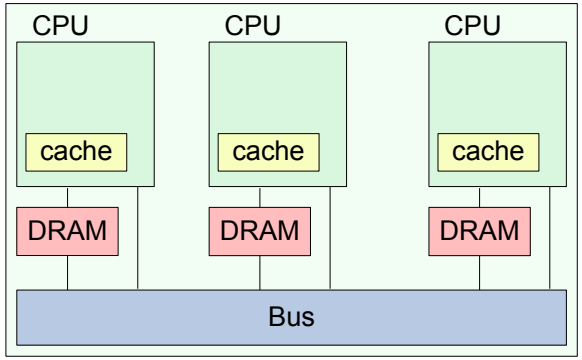
- Выбор оптимального представления графа
- По возможности организация последовательного доступа к данным
- По возможности избегать использовать межпотоковые синхронизации
- Стремиться работать не на задержке обращений к памяти, а на темпе
- Улучшение локализации
- Алгоритмические оптимизации
- Сжатие данных
- Аккуратная работа с памятью внутри NUMA-вычислительного узла
- Балансировка нагрузки
- Аккуратно измерять производительность

Архитектура вычислительных систем с распределенной памятью

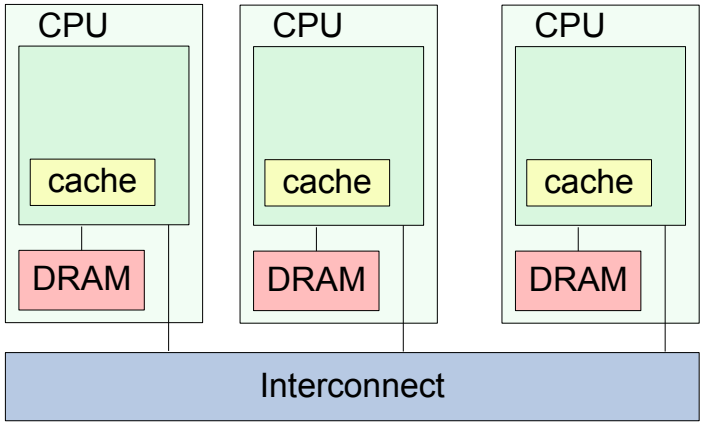
Классификация вычислительных систем

	Общее адресное пространство	**Раздельное адресное пространство**
Общая память	SMP (Symmetric Multiprocessing)	–
Распределенная память	NUMA (Non-Uniform Memory Access)	MPP (Massively Parallel Processing)

Общая память



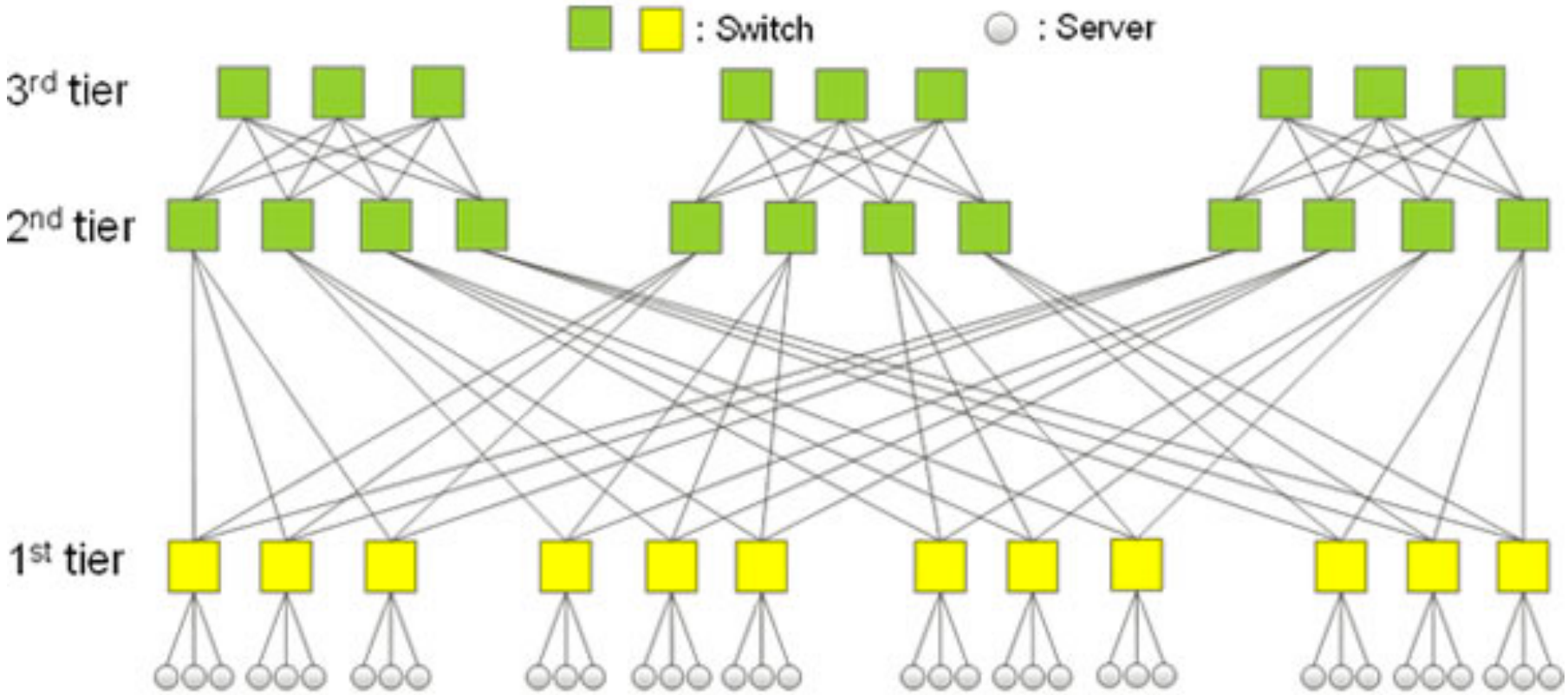
Распределенная память



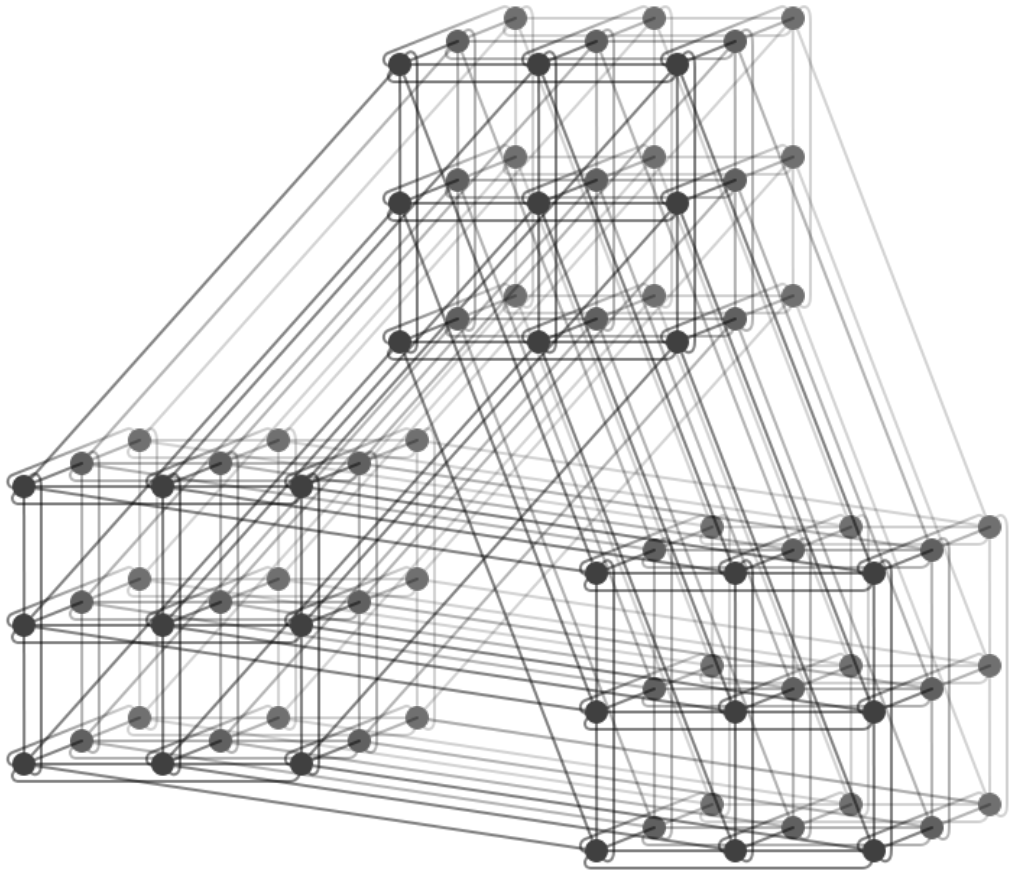
Программные модели:

- С общей памятью: pthreads, OpenMP, Intel TBB
- С распределенной памятью:
  - с глобальным доступом – SHMEM, UPC
  - передача сообщений – MPI

Топология Жирное дерево (Fat Tree), Infiniband



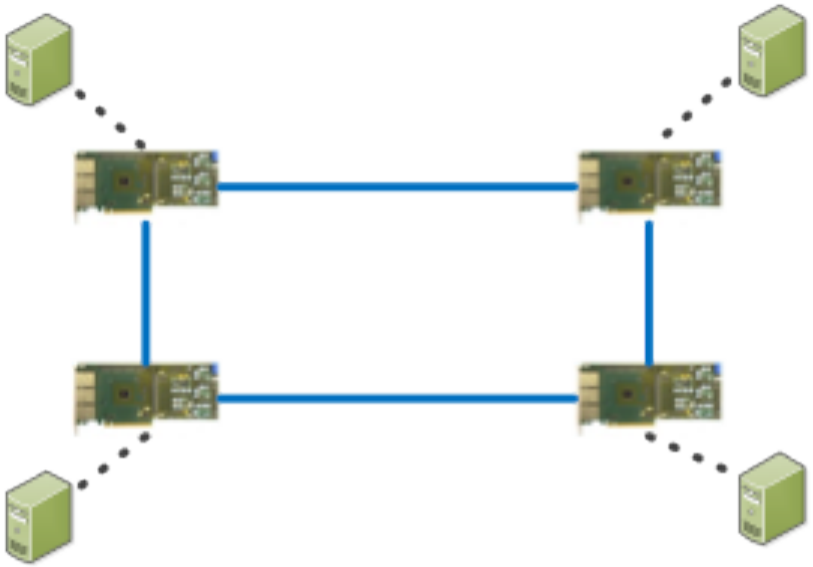
Высокоскоростная коммуникационная сеть «Ангара»



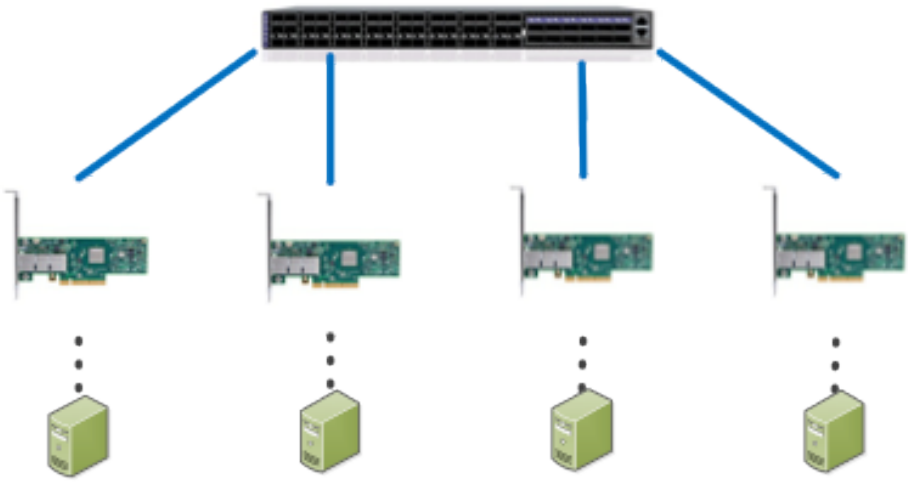
- **Топология «4D-тор»**
- **Односторонние коммуникации:**
  - put (запись в удалённую память)
  - get (чтение из удалённой памяти)
  - атомарные операции add и xor
- **Коллективные операции:**
  - broadcast
  - reduce
- **Поддержка многоядерности**
- **Адаптивная передача пакетов**
- **Механизмы синхронизации**

Тор vs Жирное дерево (Ангара vs Infiniband)

Ангара
Тор



Infiniband
Жирное дерево



Общие методы оптимизации программ для систем с распределенной памятью

Библиотека MPI

Факторы, влияющие на производительность MPI-программ

Вычислительная система

- Процессор – тип, частота, количество
- Подсистема памяти – конфигурация памяти и кэшей, пропускные способности DRAM-cache-CPU
- Сеть
- Операционная система

Коммуникационная сеть

- Топология, правила маршрутизации
- Аппаратная поддержка коллективных операций, синхронизаций
- Характеристики производительности в разных режимах работы

Приложение

- Алгоритм, его масштабируемость
- Соотношение объема коммуникаций к вычислениям
- Пространственная характеристика использования памяти и сети, размеры сообщений
- Балансировка нагрузки
- Ввод/вывод

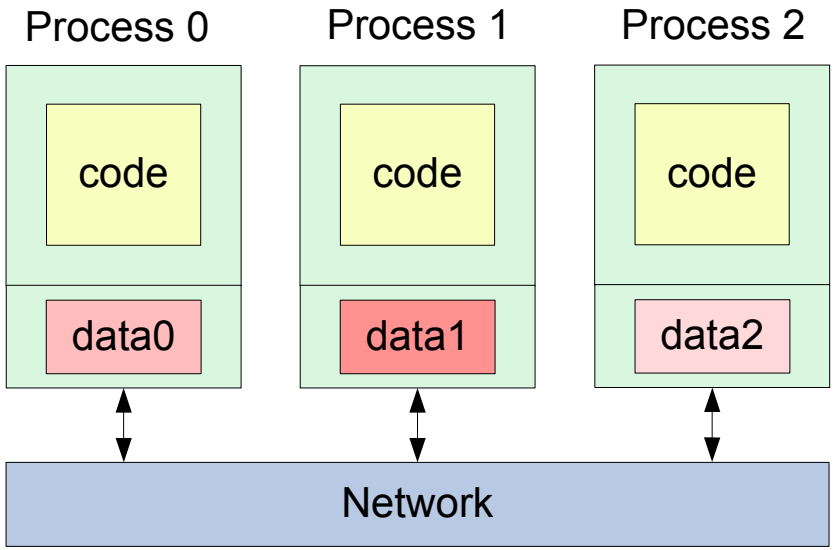
Характеристики реализации MPI и способы использования библиотеки

Message Passing Interface, MPI

- Основная цель: обеспечение переносимости программ, эффективная реализация, поддержка гетерогенных параллельных архитектур
- MPI-1 draft 1992, MPI-1.1 – 1995
- MPI-2 draft 1996, MPI-2.2 – 2009
- MPI-3 draft 2010
- Поддерживается на всех суперкомпьютерах, C, Fortran
- MPI-1 – около 125 функций, но часто используется лишь небольшое подмножество
  - Существуют эффективные реализации, используется большинством приложений
- Состав MPI-1:
  - Функции инициализации и завершения
  - Операции точка-точка
  - Коллективные операции

MPI

- Single Program Multiple Data (SPMD)
- Каждый процесс выполняет один и тот же код, но работает в своем адресном пространстве
- Все процессы порождаются во время запуска задачи, порождение новых не допускается
- Взаимодействие – при помощи отправки и приема сообщений



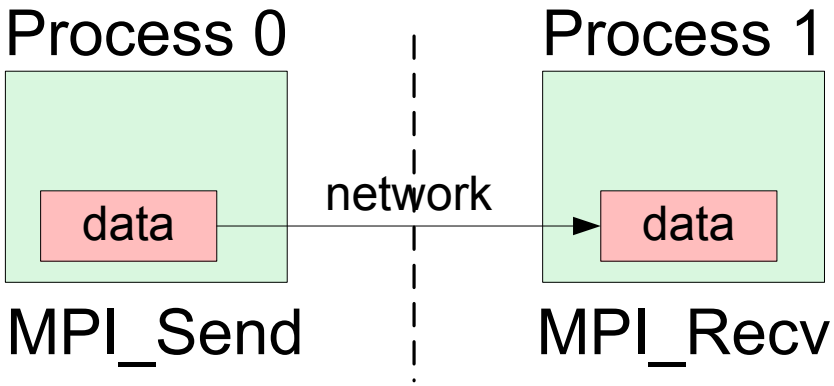
Коммуникаторы MPI

- Коммуникатор представляет группу процессов, которые могут общаться друг с другом
- Большинство функций MPI требуют в качестве аргумента коммуникатор
- ***MPI_COMM_WORLD*** собирает все MPI-процессы задачи
- Можно создавать новые коммуникаторы
- ***MPI_Comm_size*** *(comm, size, ierr)* возвращает количество процессов в коммуникаторе
- ***MPI_Comm_rank*** *(comm, rank, ierr)* возвращает номер процесса в коммуникаторе
- Чаще всего эти функции вызываются только после инициализации

```
program simple
      include ‘mpif.h’
      integer ierr, np, mype
      call MPI_Init(ierr)
      call MPI_Comm_size(MPI_COMM_WORLD, np , ierr)
      call MPI_Comm_rank(MPI_COMM_WORLD, mype, ierr)
      :
      call MPI_Finalize(ierr)
end program
```

Операции точка-точка

- ***MPI_Send (sendbuf, count, type, dest, tag, comm, ierr)*** – блокирующая посылка, возвращает значение, когда данные посланы из буфера
- ***MPI_Recv (recvbuf, count, type, source, tag, comm, status, ierr)*** – блокирующий прием, возвращает значение, когда данные получены в буфер



Тип MPI	**Тип Fortran**
MPI_INTEGER	INTEGER
MPI_REAL	REAL
MPI_DOUBLE_PRECISION	DOUBLE PRECISION
MPI_COMPLEX	COMPLEX
MPI_LOGICAL	LOGICAL
MPI_CHARACTER	CHARACTER(1)

Протоколы передачи данных в MPI: Eager

Сообщение сразу посылается на узел-получатель, но, возможно, там буферизуется

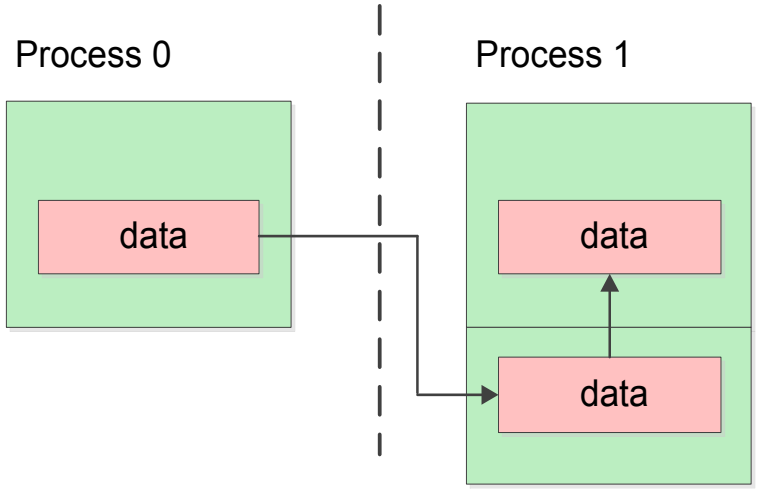
Преимущества:

- Уменьшается коммуникационная задержка

Недостатки:

- Требуется буферизация – протокол не масштабируем
- Расход памяти, даже если это не требуется
- Возможно переполнение буфера
- Может потребоваться дополнительное копирование

Используется для коротких сообщений



Протоколы передачи данных в MPI: Rendezvous

Когда нельзя предсказать состояние буфера приема или когда достигнуты ограничения eager

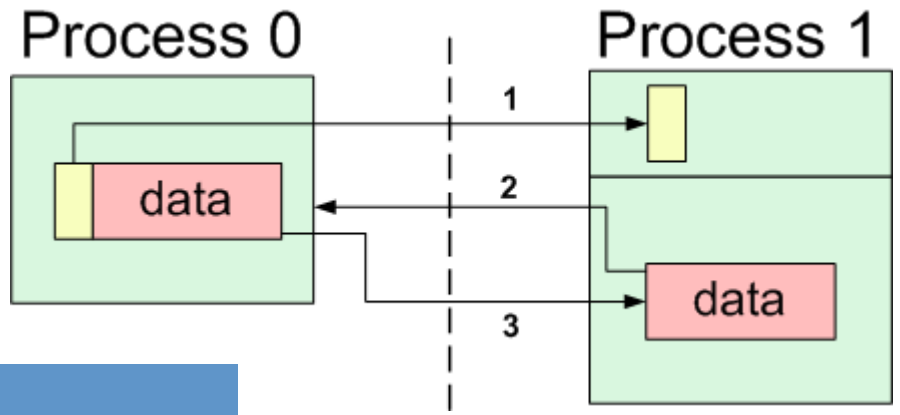
Преимущества:

- Масштабируем по сравнению с eager
- Требуется небольшой объем памяти для хранения метаданных сообщения
- Устойчивость к переполнению памяти на принимающем узле
- Отсутствие копирования данных

Недостатки:

- Растет задержка передачи сообщения из-за дополнительных коммуникаций

Используется для длинных сообщений



Переменные окружения:

- EAGER_LIMIT
- BUFFER_MEM

Эффективность программ:

Выдавать Recv раньше Send