Лабораторна робота №5. Використання кеші-пам'яті мікропроцесора

1 Мета роботи

Проаналізувати роботу різних рівнів кеш-пам'яті мікропроцесорів сімейства x86 в різних умовах.

2 Теоретичні відомості

Рекомендована література та джерела інформації:

1. Крис Касперски "Техника оптимизации программ: эффективное использование памяти", глава 3;
2. Ulrich Drepper, "What Every Programmer Should Know About Memory", http://people.redhat.com/drepper/cpumemory.pdf;
3. Intel® 64 and IA-32 Architectures Optimization Reference Manual, http://developer.intel.com/design/processor/manuals/248966.pdf;
4. Software Optimization Guide for AMD64 Processors, http://www.amd.com/us-en/assets/content_type/white_papers_and_tech_docs/25112.PDF.

3 Методичні вказівки

Для виконання завдання даної лабораторної роботи найбільш доцільним є використання мови C. Завдання вимагають від вас визначення швидкості обробки елементу даних з блоку пам'яті певного розміру. Оскільки дана робота не вимогає від вимірювань надвеликої точності, то для визначення часу обробки окремого елементу можна використовувати наступну формулу:

$$T_{element} = \frac{T_{block}}{N_{elements}}$$

де $T_{element}$ – час обробки окремого елементу,
T_{block} – час обробки всього блоку,
$N_{elements}$– загальна кількість елементів, що оброблюється.

Для визначення T_{block} вам знадобиться таймер, що може достатньо точно вимірювати інтервали часу.

- Під ОС Linux (та іншими POSIX-сумісними операційними системами) для цього можна використовувати функцію clock_gettime (функція об'явлена у файлі time.h). У більшості випадків для використання цієї функції необхідно приєднати до вашої програми системну бібліотеку **librt**. Це можна зробити за допомогою ключа -lrt компілятора gcc.
- Під ОС Windows для цього можна використовувати функції QueryPerformanceCounter та QueryPerformanceFrequency (функції об'явлені у файлі Windows.h). У додатку А нижче наводиться приклад використання цих функцій для вимірювання часу.

Нижче наводиться приклад основного робочого циклу програми для вимірювання **у випадку операції читання-запису**.

```
int* p = (int*)malloc(MAX_BLOCK_SIZE);
int tmp = 0;
for (int b=MIN_BLOCK_SIZE; b < MAX_BLOCK_SIZE; b += STEP) {
  for (int c = 0; c <= b; c += sizeof(int)) {
    tmp += *(int*)((int)p + c);
    *(int*)((int)p + c) = tmp;
  }
}
```

Для виконання завдання лабораторної роботи вам необхідно час виконання однієї ітерації внутрішнього циклу в залежності від розміру блоку, що оброблюється в ньому.

4 Порядок виконання роботи

1. Проаналізувати умови індивідуального завдання;
2. Визначити розміри кеш-пам'яті процесора, на якому будуть проводитися вимірювання;
3. Розробити програму для виконання необхідних вимірювань;
4. Отримані результати необхідно представити у графічному вигляді;
5. Результати додаткових завдань мають бути представлені після основного завдання.

Вимоги до графічного представлення результатів

Оскільки значна частина захисту роботи буде полягати у обговоренні отриманих вами результатів, то форма їх представлення не повинна заважати цьому. Графік, що ви побудуєте, має бути контрастним, мати масштаб, достатній для розглядання деталей та цифрові шкали. **Невиконання цих вимог призведе до суттєвого зниження оцінки.**

5 Завдання

В усіх завданнях необхідно виміряти швидкість обробки елементів блоку пам'яті в залежності від його розміру при різних умовах. В завданнях використовуються наступні умовні позначення:

- L1.size – розмір кеш-пам'яті першого рівня;
- L2.size – розмір кеш-пам'яті другого рівня.

В якості кроку інкременту розміру блоку достатньо взяти 1 кілобайт.

Варіанти індивідуальних завдань

1. Дослідити швидкість обробки до елементів блоку пам'яті при виконанні операції читання. Розмір блоку пам'яті повинен змінюватись від 10% від L1.size до 200% від L1.size;
2. Дослідити швидкість обробки до елементів блоку пам'яті при виконанні операції запису. Розмір блоку пам'яті повинен змінюватись від 10% від L1.size до 200% від L1.size;
3. Дослідити швидкість обробки до елементів блоку пам'яті при виконанні операцій читання-запису. Розмір блоку пам'яті повинен змінюватись від 10% від L1.size до 200% від L1.size;
4. Дослідити швидкість обробки до елементів блоку пам'яті при виконанні операцій запису-читання. Розмір блоку пам'яті повинен змінюватись від 10% від L1.size до 200% від L1.size;
5. Дослідити швидкість обробки до елементів блоку пам'яті при виконанні операції читання. Розмір блоку пам'яті повинен змінюватись від 10% від L2.size до 200% від L2.size;
6. Дослідити швидкість обробки до елементів блоку пам'яті при виконанні операції запису. Розмір блоку пам'яті повинен змінюватись від 10% від L2.size до 200% від L2.size;
7. Дослідити швидкість обробки до елементів блоку пам'яті при виконанні операцій читання-запису. Розмір блоку пам'яті повинен змінюватись від 10% від L2.size до 200% від L2.size;
8. Дослідити швидкість обробки до елементів блоку пам'яті при виконанні операцій запису-читання. Розмір блоку пам'яті повинен змінюватись від 10% від L2.size до 200% від L2.size;

Додаткові завдання

1. Оцінити похибки вимірювань;
2. Реалізувати аналогічний алгоритм вимірювання на мові високого рівня (Java, Python, Ruby і т.п.). Порівняти результати вимірювань.

6 Контрольні запитання

1. Яке призначення кеш-пам'яті?
2. Чим відрізняються різні рівні кеш-пам'яті?
3. Поняття кеш-лінійки (строки), тегу лінійки, асоціативної пам'яті.
4. Стратегії заміщення в кеш-пам'яті: LRU, LFU, FIFO. Які їх принципи роботи, переваги та недоліки?
5. Стратегії завантаження даних у кеш-пам'ять: on-demand, speculative. Які їх відмінності, переваги та недоліки?
6. Поняття асоціативності кеш-пам'яті. Які є переваги та недоліки у великого ступеня асоціативності?
7. Політики запису у кеш-пам'яті: Write Through (WT), Write Combining (WC), Write Back (WB). Які їх відмінності?
8. Буфери запису процесора. Яке їх призначення та механізм роботи?

Додаток А

Вимірювання часу на платформі Windows

```
typedef struct prof_timer_t {
  LARGE_INTEGER time_start;
  LARGE_INTEGER time_stop;
} prof_timer_t;
void prof_timer_start(prof_timer_t *timer) {
  QueryPerformanceCounter(&timer->time_start);
}
void prof_timer_stop(prof_timer_t *timer) {
  QueryPerformanceCounter(&timer->time_stop);
}
double prof_timer_get_duration_in_secs(prof_timer_t *timer) {
  LARGE_INTEGER freq;
  QueryPerformanceFrequency(&freq);
  double duration = (double)(timer->time_stop.QuadPart - timer->time_start.QuadPart);
  double scaled_duration = duration/(double)freq.QuadPart;
  return scaled_duration;
}
```