[CSEG437/CSE5437] 수치 컴퓨팅 및 GPU 프로그래밍 (Numerical Computing and GPU Programming)

서강대학교 공과대학 컴퓨터공학과
교수 임 인 성

강의 소개

- 과목명: **수치 컴퓨팅 및 GPU 프로그래밍**
  (Numerical Computing and GPU Programming)
- 과목 번호 및 분반: **CSEG437/CSE5437**
- 강의실 및 강의 시간: **AS-303, 월12:00~13:15/금10:30~11:45**
- 선수 지식: **대학 수준의 미적분과 선형대수, 4학년 수준의 C/C++ 프로그래밍**

- 담당 교수: **임 인 성** (AS-905, ihm@sogang.ac.kr)
- 담당 조교: **안 재 풍** (AS-914, ajp5050@sogang.ac.kr)

- **과목 홈페이지:** http://grmanet.sogang.ac.kr에서 찾을 것.
  - **조교가 수시로 중요 공지 사항을 본 과목 홈페이지에 게시하거나 이메일을 발송할 예정임**.
  - 본인의 책임 하에 수시로 확인할 것.

- **주의:** 본 과목은 고급 소프트웨어 실습 I 과목에서 배운 다음 주제에 대하여 이해하고 있다고 가정함.
  - Floating-point numbers and operations
  - Solving nonlinear equations and systems of nonlinear equations
  - Numerical integrations
  - Generation of random sample numbers from any probability distribution
  - Calling Fortran functions from C/C++ programs
  - High-level code optimizations
  - Basic CUDA programming

교과 목표: CSE4140 수치 컴퓨팅 및 응용

- 컴퓨터학 및 여러 공학 분야에서 효과적인 수치 컴퓨팅(numerical computing)을 필요로 하는 문제들을 자주 볼 수가 있다. 예를 들어 컴퓨터 시스템의 성능을 평가하거나 방대한 웹 페이지의 탐색을 위해서 풀어야 하는 선형 방정식 문제, 자연스러운 컴퓨터 애니메이션을 제작하기 위하여 풀어야 하는 미분 방정식 문제, 멀티미디어 데이터의 효과적인 처리를 위한 행렬과 벡터 계산 등은 일반적으로 컴퓨터학에서 배우는 이산 알고리즘(discrete algorithm)과는 본질적으로 다른 해법을 필요로 한다. 특히 이러한 문제들은 C 언어의 int나 char와 같은 타입의 정수 연산이 아니라, float와 double과 같은 타입의 부동 소수점 연산을 통하여 해결해야 하기 때문에, 무한개가 존재하는 실수를 유한개의 부동 소수점 숫자로 효과적으로 처리할 수 있는 기법에 대한 이해가 필요하다. 본 과목에서는 다음과 같은 두 가지 문제를 집중적으로 다룬다.

- **첫째**, 부동 소수점 연산에 기반을 둔 수치 계산을 부주의하게 프로그래밍 할 경우 발생할 수 있는 문제점을 파악한 후, 그러한 문제들을 해결하는데 필요한 기법에 대하여 알아보고, 이와 함께 다양한 형태의 프로세서 상에서의 코드 최적화 기법에 대해서도 살펴본다.
- **둘째**, 다양한 실제 문제를 수학적으로 모델링을 한 후, 이들을 해결하는데 필요한 풀이 기법에 대하여 알아본다. 수업 시간에 익힌 다양한 수치 해석 이론을 실제 문제 해결에 적용할 수 있는 능력을 배양하기 위하여 C/C++ 및 수치해석 관련 공개 소프트웨어들을 이용하여 4-5 차례의 프로그래밍 프로젝트를 수행하도록 한다.

교과 목표

- 본 과목에서는 대략적으로 전체 강의 시간을 2등분 하여 다음 내용에 대하여 학습한다.

- **첫째,** 컴퓨터상에서의 수치 계산의 특성을 이해한 후, 몇 가지 주제를 선정하여 관련 수치 알고리즘을 이해하고 실제로 구현하여 본다.

- **둘째,** OpenCL API를 기반으로 many-core processor인 GPU 상에서의 효과적인 프로그래밍 기법을 익힌다. 이를 위하여 몇 가지 병렬 알고리즘에 대하여 배우고, 주어진 수치 계산 문제를 GPU를 사용하여 가속하여 본다.

Why Numerical Computing?

- Computing in continuous space
  - Discrete space versus continuous space
- Computing using floating-point arithmetic
  - Fixed-point arithmetic versus floating-point arithmetic
- Problems in real-world
  - Mathematical formulations
  - Numerical methods
  - Real numbers on "discrete" machine
- High computational expense
  - Code optimization
  - Parallel processing
    - CUDA and OpenCL on GPU

By courtesy of D. N. Arnold
(http://www.ima.umn.edu/~arnold/disasters/patriot.html)

The Patriot Missile Failure

On February 25, 1991, during the Gulf War, an American Patriot Missile battery in Dharan, Saudi Arabia, **failed to track and intercept an incoming Iraqi Scud missile**. The Scud struck an American Army barracks, killing 28 soldiers and injuring around 100 other people. ... It turns out that **the cause was an inaccurate calculation of the time since boot due to computer arithmetic errors**. Specifically, the time in tenths of second as measured by the system's internal clock **was multiplied by 1/10** to produce the time in seconds. This calculation was performed using **a 24 bit fixed point register**. In particular, **the value 1/10, which has a non-terminating binary expansion**, was chopped at 24 bits after the radix point. **The small chopping error, when multiplied by the large number giving the time in tenths of a second, led to a significant error**. Indeed, the Patriot battery had been up around 100 hours, and an easy calculation shows that **the resulting time error due to the magnified chopping error was about 0.34 seconds**. (The number 1/10 equals $1/2^4+1/2^5+1/2^8+1/2^9+1/2^{12}+1/2^{13}+\ldots$. In other words, the binary expansion of 1/10 is 0.0001100110011001100110011001100.... Now the 24 bit register in the Patriot stored instead 0.000110011001100110011001 introducing an error of 0.0000000000000000000000011001100... binary, or about 0.000000095 decimal. Multiplying by the number of tenths of a second in 100 hours gives $0.000000095\times100\times60\times60\times10=0.34$.) A Scud travels at about 1,676 meters per second, and so travels more than half a kilometer in this time. This was far enough that the incoming Scud was outside the "range gate" that the Patriot tracked. **Ironically, the fact that the bad time calculation had been improved in some parts of the code, but not all, contributed to the problem, since it meant that the inaccuracies did not cancel.**

왜 전문적인 S/W 개발 시 수치 컴퓨팅에 대한 이해가 필요한가?

Problems with Floating-Point Arithmetic

- **CPU에 따른 실수에 대한 저장 값 차이**

  - 왜 컴퓨터는 1.1과 같은 단순한 숫자조차 정확하게 표현을 하지 못할까?
  - 왜 사용하는 CPU에 따라 동일한 실수에 대하여 실제로 저장한 내용이 다를까?

- **컴파일러 옵션에 따른 부동 소수점 연산 결과의 불확실성**

  - **Disable(Debug) 옵션 vs Maximize Speed 옵션**

    - 소프트웨어 개발 시 디버그 모드에서 정확한 수치 계산 결과를 확인한 후, 최종적으로 컴파일러 옵션을 사용하여 코드를 최적화하여 수행하면 수치적으로 다른 결과 값이 나오는 경험을 해본 적이 있는가?

- **이차 방정식에 대한 근의 공식의 계산**
  - **중학교 수학 시간에 배운 바에 의하면,**

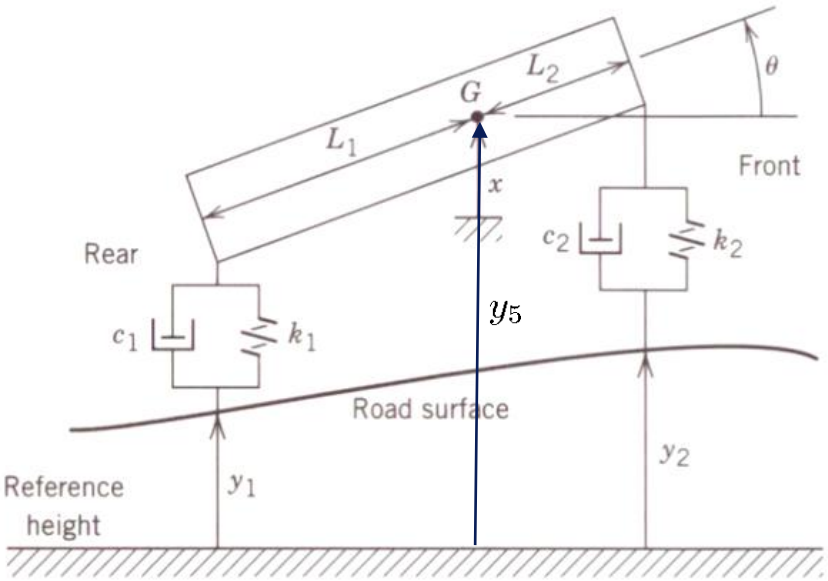
$$0.5x^2 + bx + c = 0 \longrightarrow x = -b \pm \sqrt{b^2 - 2c}$$

  - **이 중 값이 큰 근에 대하여,**

$$-b + \sqrt{b^2 - 2c} = \frac{-2c}{b + \sqrt{b^2 - 2c}}$$

- 컴퓨터를 사용하여 위 식의 각 변의 값을 계산할 경우, 어떤 *b*, *c* 값에 대해서는 양변의 값이 서로 달라지는데 그 이유는 무엇일까?
- 이 두 식 중 어떤 식을 사용하는 것이 수치적으로 더 안전할까?
- 왜 왼쪽의 수식을 어떻게 프로그래밍하는가에 따라 그 결과가 다르게 나오곤 할까?

3D Games and Differential Equations



2nd-order ODE system

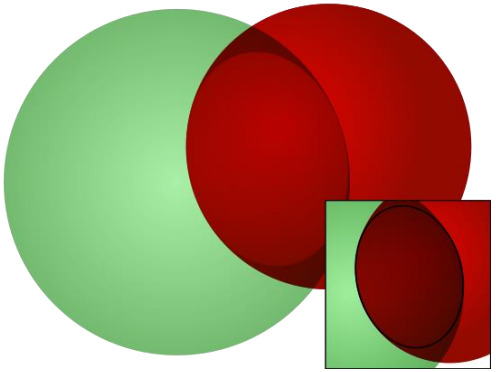
$$
\begin{aligned}
m \cdot y_5'' &= -(k_1+k_2)y_5 - (c_1+c_2)y_5' + (k_1L_1 - k_2(L-L_1))\theta \\
&\quad + (c_1L_1 - c_2(L-L1))\theta' + (k_1y_1 + k_2y_2 + c_1y_1' + c_2y_2') \\
I \cdot \theta'' &= (L_1k_1 - (L-L_1)k_2)y_5 + (L_1c_1 - (L-L_1)c_2)y_5' + \\
&\quad - (L_1^2k_1 + (L-L_1)^2k_2)\theta - (L_1^2c_1 + (L-L_1)^2c_2)\theta' \\
&\quad + (-L_1k_1y_1 + (L-L_1)k_2y_2 - L_1c_1y_1' + (L-L_1)c_2y_2')
\end{aligned}
$$

Global Positioning System Equations

Find (x, y, z) such that

$$\begin{aligned}
(x-a_1)^2+(y-b_1)^2+(z-c_1)^2 &= [(C(tr_1+b-t_1)]^2 \\
(x-a_2)^2+(y-b_2)^2+(z-c_2)^2 &= [(C(tr_2+b-t_2)]^2 \\
(x-a_3)^2+(y-b_3)^2+(z-c_3)^2 &= [(C(tr_3+b-t_3)]^2 \\
(x-a_4)^2+(y-b_4)^2+(z-c_4)^2 &= [(C(tr_4+b-t_4)]^2,
\end{aligned}$$

where, for each satellite i, the locations are (a_i, b_i, c_i), tr_i is the indicated time the message was received by the receiver, b is the clock error or bias, t_i is synchronized transmission time from the satellite, and C is the speed of light.



From Wikipedia

Shrek and Differential Equations

Conservation of momentum

$$\frac{\partial \mathbf{u}}{\partial t} = \nu \nabla \cdot (\nabla \mathbf{u}) - (\mathbf{u} \cdot \nabla)\mathbf{u} - \frac{1}{\rho}\nabla p + \mathbf{g}$$

viscosity convection pressure

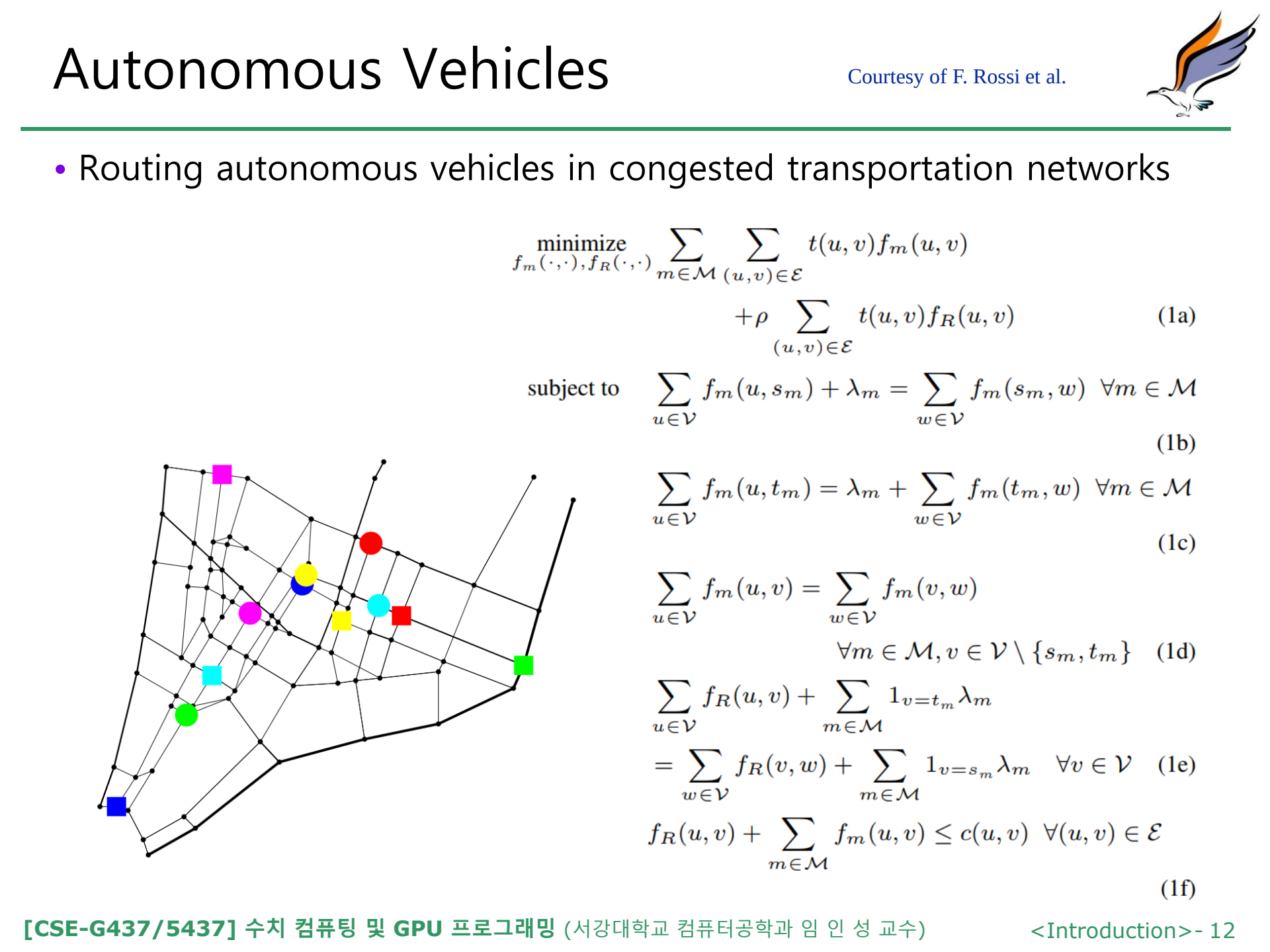
Conservation of mass

$$\nabla \cdot \mathbf{u} = 0$$

$$\nabla = \{\partial / \partial x, \partial / \partial y, \partial / \partial z\}$$

Autonomous Vehicles

Courtesy of F. Rossi et al.



- Routing autonomous vehicles in congested transportation networks

$$\underset{f_m(\cdot,\cdot),f_R(\cdot,\cdot)}{\text{minimize}} \sum_{m\in\mathcal{M}} \sum_{(u,v)\in\mathcal{E}} t(u,v) f_m(u,v)$$

$$+\rho \sum_{(u,v)\in\mathcal{E}} t(u,v) f_R(u,v) \tag{1a}$$

$$\text{subject to} \quad \sum_{u\in\mathcal{V}} f_m(u,s_m) + \lambda_m = \sum_{w\in\mathcal{V}} f_m(s_m,w) \quad \forall m \in \mathcal{M} \tag{1b}$$

$$\sum_{u\in\mathcal{V}} f_m(u,t_m) = \lambda_m + \sum_{w\in\mathcal{V}} f_m(t_m,w) \quad \forall m \in \mathcal{M} \tag{1c}$$

$$\sum_{u\in\mathcal{V}} f_m(u,v) = \sum_{w\in\mathcal{V}} f_m(v,w)$$

$$\forall m \in \mathcal{M}, v \in \mathcal{V} \setminus \{s_m,t_m\} \tag{1d}$$

$$\sum_{u\in\mathcal{V}} f_R(u,v) + \sum_{m\in\mathcal{M}} 1_{v=t_m}\lambda_m$$

$$= \sum_{w\in\mathcal{V}} f_R(v,w) + \sum_{m\in\mathcal{M}} 1_{v=s_m}\lambda_m \quad \forall v \in \mathcal{V} \tag{1e}$$

$$f_R(u,v) + \sum_{m\in\mathcal{M}} f_m(u,v) \le c(u,v) \quad \forall (u,v) \in \mathcal{E} \tag{1f}$$

Data Mining and Matrix Computation

- Problem: **Search books about *Baking Bread.***

The $t = 6$ **terms**:

T1: bak(e,ing)
T2: recipes
T3: bread
T4: cake
T5: pastr(y,ies)
T6: pie

$$A_k^t q = (V_k D_k)(U_k^t q)$$

The $d = 5$ **document** titles:

D1: How to Bake Bread Without Recipes
D2: The Classic Art of Viennese Pastry
D3: Numerical Recipes: The Art of Scientific Computing
D4: Breads, Pastries, Pies and Cakes: Quantity Baking Recipes
D5: Pastry: A Book of Best French Recipes

왜 전문적인 S/W 개발 시 수치 컴퓨팅에 대한 이해가 필요한가?

Mathematical Computing on Low-End Processors

- Find the base-two logarithm $s = \log_2(n)$ for a 32-bit unsigned integer n.

  $\longrightarrow$ Find a Q26 representation q so that $q = s2^{26}$.

  - Calculate the integer part of s using CLZ.

On ARM processor

  - Reduce the number to the range $1 \leq n < 2$.

  - Perform a table lookup on an approximation a to n to find $\log_2(a)$ and $\frac{1}{a}$.

$$\log_2(n) = \log_2(a) + \log_2(\tfrac{n}{a}) \text{ where } a \approx n.$$

  - Find $\log_2(\frac{n}{a})$ to improve the result accuracy.

$$\log_2(1+x) = \frac{\log_e(1+x)}{\log_e(2)} = \frac{1}{\log_2(2)}(x - \frac{x^2}{2} + \frac{x^3}{3} - \cdots) \text{ where } x \approx 0.$$

- The following code uses 31 cycles on an ARM9E processor (accurate to an error of 2^{-25}).

```
q       RN 3
t       RN 12

        ; int ulog2_32(unsigned n)
ulog2_32
        CLZ     r, n
        MOV     d, n, LSL#1
        MOV     d, d, LSL r          ; 1<=d<2 at Q32
        RSB     n, r, #31            ; integer part of the log
        MOV     r, d, LSR#27         ; estimate e=1+(r/32)+(1/64)
        ADR     t, ulog2_table
        LDR     r, [t, r, LSL#3]!    ; r=log2(e) at Q26
        LDR     q, [t, #4]           ; q=1/e     at Q32
        MOV     t, #0
        UMLAL   t, r, d, r           ; r=(d/e)-1 at Q32
        LDR     t, =0x55555555       ; round(2^32/3)
        ADD     n, q, n, LSL#26      ; n+log2(e) at Q26
        SMULL   t, q, r, t           ; q = r/3   at Q32
        LDR     d, =0x05c551d9       ; round(2^26/ln(2))
        SMULL   t, q, r, q           ; q = r^2/3 at Q32
        MOV     t, #0
        SUB     q, q, r, ASR#1       ; q = -r/2+r^2/3 at Q32
        SMLAL   t, r, q, r           ; r - r^2/2 + r^3/3 at Q32
        MOV     t, #0
        SMLAL   t, n, d, r           ; n += r/log(2) at Q26
        MOV     pc, lr
```

From "ARM System Developer's Guide" by Sloss et al. (2004)

```
ulog2_table
        ; table[2*i]   =round(2^32/a)       where a=1+(i+0.5)/32
        ; table[2*i+1]=round(2^26*log2(a)) and 0<=i<32
        DCD 0xfc0fc0fc, 0x0016e797, 0xf4898d60, 0x0043ace2
        DCD 0xed7303b6, 0x006f2109, 0xe6c2b448, 0x0099574f
        DCD 0xe070381c, 0x00c2615f, 0xda740da7, 0x00ea4f72
        DCD 0xd4c77b03, 0x0111307e, 0xcf6474a9, 0x0137124d
        DCD 0xca4587e7, 0x015c01a4, 0xc565c87b, 0x01800a56
        DCD 0xc0c0c0c1, 0x01a33761, 0xbc52640c, 0x01c592fb
        DCD 0xb81702e0, 0x01e726aa, 0xb40b40b4, 0x0207fb51
        DCD 0xb02c0b03, 0x0228193f, 0xac769184, 0x0247883b
        DCD 0xa8e83f57, 0x02664f8d, 0xa57eb503, 0x02847610
        DCD 0xa237c32b, 0x02a20231, 0x9f1165e7, 0x02bef9ff
        DCD 0x9c09c09c, 0x02db632d, 0x991f1a51, 0x02f7431f
        DCD 0x964fda6c, 0x03129ee9, 0x939a85c4, 0x032d7b5a
        DCD 0x90fdbc09, 0x0347dcfe, 0x8e78356d, 0x0361c825
        DCD 0x8c08c08c, 0x037b40e4, 0x89ae408a, 0x03944b1c
        DCD 0x8767ab5f, 0x03acea7c, 0x85340853, 0x03c52286
        DCD 0x83126e98, 0x03dcf68e, 0x81020408, 0x03f469c2
```

Efficient Programming on ARM Processors

```
int checksum_v (int *data) {
  char i;
  int sum = 0;

  for (i = 0; i < 64; i++) {
    sum += data[i];
  }
  return sum;
}
```

V.S.

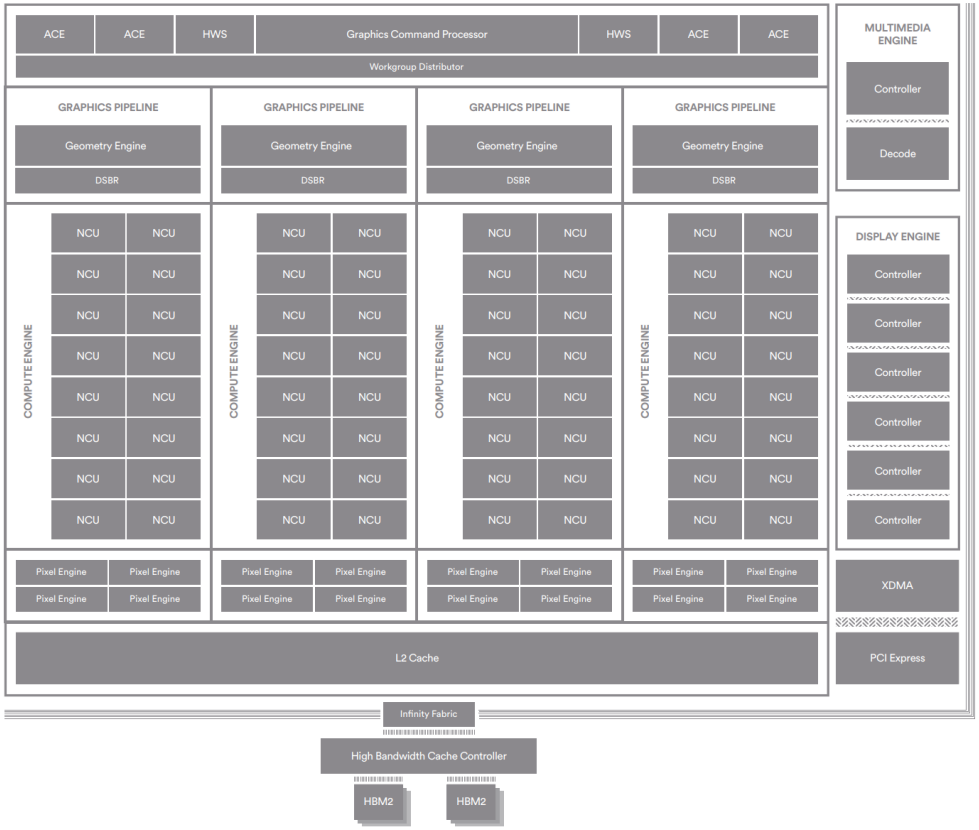
```
short checksum_v (short *data) {
  unsigned int i;
  int sum = 0;

  for (i = 0; i < 64; i++) {
    sum += *(data++);
  }
  return (short)sum;
}
```

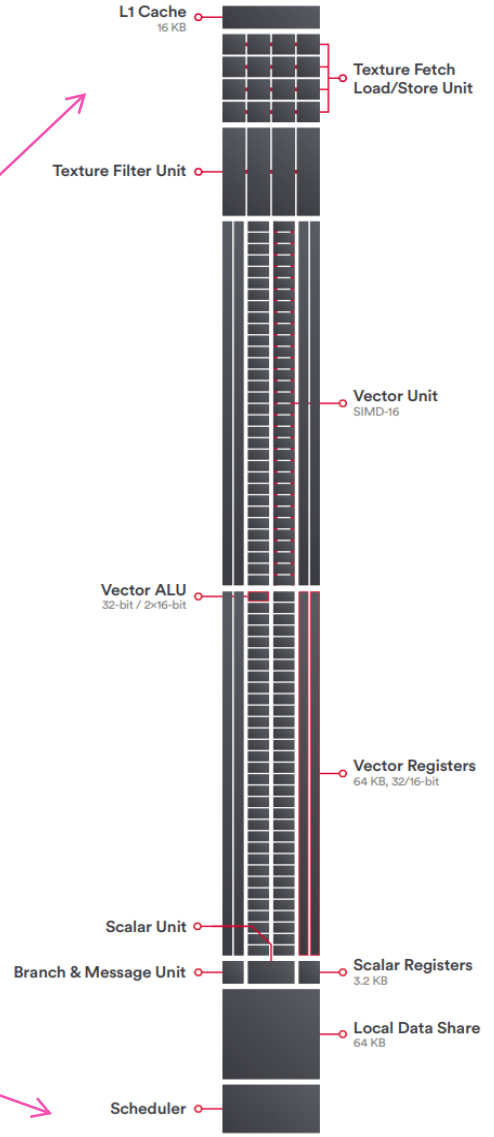
왜 전문적인 S/W 개발 시 수치 컴퓨팅에 대한 이해가 필요한가?

Parallel Computing on GPU

1999	Intel ASCI Red/9632	2.3796 TFLOPS	New Mexico, USA
2000	IBM ASCI White	7.226 TFLOPS	DoE-Lawrence Livermore National Laboratory, California, USA
2002	NEC Earth Simulator	35.86 TFLOPS	Earth Simulator Center, Yokohama, Japan
2004	IBM Blue Gene/L	70.72 TFLOPS	DoE/IBM Rochester, Minnesota, USA
2005		136.8 TFLOPS	DoE/U.S. National Nuclear Security Administration, Lawrence Livermore National
		280.6 TFLOPS	

- **AMD Radeon RX Vega64 Liquid Cooled Edition**
  - **64 next-generation compute units (NCUs)**
  - **Each NCU has 64 stream processors.**
  - ➢ **A total of 4,096 stream processors**
  - ➢ **13.7 TFLOPS**





© AMD

OpenCL versus CUDA

- **OpenCL** (**Open Computing Language**), an open standard maintained by the non-profit technology consortium Khronos Group, is a framework **for writing programs that execute across heterogeneous platforms consisting of central processing units (CPUs), graphics processing units(GPUs), digital signal processors (DSPs), field-programmable gate arrays (FPGAs) and other processors or hardware accelerators.**
- **CUDA** is a parallel computing platform and application programming interface (API) model created by Nvidia. It allows software developers and software engineers to **use a CUDA-enabled graphics processing unit (GPU) for general purpose processing** – an approach termed GPGPU (General-Purpose computing on Graphics Processing Units). The CUDA platform is a software layer that gives direct access to the GPU's virtual instruction set and parallel computational elements, for the execution of compute kernels.

CSE4140 수치 컴퓨팅 및 응용 강의 내용

1) **Floating-point arithmetic**

- IEEE Binary Floating Point Arithmetic Standard 754-1985의 이해를 통하여 컴퓨터상에서의 실수 표현 방법에 대하여 알아본다.
- 유한개의 비트를 사용하여 무한개의 실수를 표현할 때의 어떤 부류의 실수들이 표현이 되는지와 표현 오차를 비롯한 여러 문제점에 대하여 살펴본다.

2) **Numerical problems with floating-point arithmetic**

- C 언어의 float 또는 double 타입의 수와 같은 floating-point number를 이용하여 연산을 수행할 때 발생할 수 있는 문제점에 대하여 이해한다. 예를 들어 실수 계산을 할 경우 컴퓨터상에서는 왜 덧셈의 교환 법칙이 성립하지 않는지, 비슷한 숫자 간의 뺄셈이 얼마나 위험한지, 같은 C 코드에 대해서도 컴파일러 옵션의 선택에 따라 왜 계산 결과가 달라지는지 등의 여러 문제를 다룬다.
- 그러한 문제를 해결하기 위한 프로그래밍 기법들에 대하여 알아본다.

3) **Stability of numerical algorithms**

- 수치 계산에 기반을 둔 알고리즘은 그 계산 방식에 따라 수치적인 안정성에 큰 차이를 보이는데, 다양한 예를 통하여 수치 알고리즘의 이러한 측면을 이해한다.
- 주어진 수치 문제를 안정적으로 계산해주는 알고리즘을 개발하는데 있어 고려할 점에 대하여 배운다.

4) **Fixed-point arithmetic and programming on low-end processor**

- 부동 소수점 연산을 하드웨어적으로 가속해주지 못하는 저급한 CPU 상에서 실수 연산을 fixed-point arithmetic을 사용하여 시뮬레이션 해주는 기법은 과거의 기법으로 간주되어 왔으나, 최근 ARM 프로세서 등의 낮은 성능의 프로세서에 기반을 둔 장치 상에서 다양한 응용소프트웨어를 개발하려함에 따라 이 주제는 다시 중요한 문제로 대두되고 있다. 이러한 추세에 대처하기 위하여 기본적으로 정수 연산을 통하여 실수 연산을 처리해주는 기법에 대하여 이해를 해본다.

- ARM 프로세서와 같이 부동 소수점 연산이 매우 느린 저급한 프로세서 상에서의 실수 연산의 시뮬레이션을 위한 프로그래밍 기법에 대하여 익힌다.

5) **Code optimization techniques**

- 최적의 소프트웨어를 개발하는데 필요한 원시 코드 레벨에서의 최적화 기법에 대하여 알아본다.

- 멀티미디어 처리 소프트웨어의 최적화에 유용한 Intel Pentium4 CPU의 SSE 기능이 제공하는 어셈블러를 이용한 코드 최적화 기법에 대하여 이해해본다.

- ARM 프로세서와 같은 낮은 성능의 프로세서 상에서의 저수준의 코드 최적화 기법에 대하여 알아본다.

- Microsoft Visual C++의 profiling 툴, Intel의 VTune 최적화 툴, Microsoft의 Visual C++를 위한 Intel의 Optimizing C++ 컴파일러 플러그인 등에 대하여 간략히 살펴본다.

6) **Data interpolation and applications**

- 샘플링된 이산 데이터로부터 원래의 연속 정보를 복원/추정해주는 데이터 보간 기법에 대하여 살펴본다.

- 1차, 2차, 그리고 3차 보간 등 다항식을 이용한 데이터 보간 기법에 대하여 살펴본 후 실습을 통하여 실제 문제에 대하여 보간 기법을 적용하여 본다.

7) **Differential equations and applications I & II**

- 자연 현상을 모델링 하는데 있어 가장 중요한 수학적인 도구 중의 하나인 미분 방정식의 해법에 대하여 이해를 해본다.

- Euler 방법과 Runge-Kutta 방법을 비롯한 여러 풀이 기법에 대하여 살펴본 후, 각 방법의 장단점에 대하여 살펴본다.

8) **Problem solving with public domain software I**

- 수치 계산 분야의 경우 다양한 주제의 문제를 해결해주는 함수들이 많이 공개되어 있는데, 이중 Fortran 또는 C 언어로 작성되어 있는 공개 소프트웨어를 이용하여 효과적으로 문제를 해결할 수 있는 능력을 키운다(Fortran 언어에 대한 지식은 필요 없음).

- 실제로 공학 분야에서 부닥치는 응용문제 중 미분 방정식에 관련한 문제를 선정하여 공개 소프트웨어를 사용하여 문제를 풀어본다.

9) **Systems of linear equations and applications I & II**

- 선형 방정식은 공학 분야에서 가장 기초가 되는 주제일 뿐만 아니라, 이와 관련된 기본 개념은 컴퓨터 시스템의 성능 평가, 방대한 웹 페이지의 탐색과 같은 데이터 마이닝, 멀티미디어 데이터의 효과적인 처리 등의 컴퓨터학 분야에서 중요하게 적용이 되고 있다.

- 이러한 응용문제의 해결 능력을 키우기 위하여 선형 방정식의 기본 개념에 대하여 이해한 후 풀이 기법을 익히도록 한다.

- 특히 행렬 분해를 통한 직접 풀이 방법과 반복 계산을 통한 반복 기법에 대하여 살펴본 후, 각 방법의 장단점 및 적용 범위에 대하여 이해한다.

- 이러한 과정을 통하여 컴퓨터를 이용한 선형 방정식의 풀이가 경우에 따라 얼마나 부정확할 수도 있는지, 그리고 그러한 문제를 해결하기 위하여 프로그래머로서 어떠한 노력을 기울여야 할 지에 대하여 알아본다.

10) **Problem solving with public domain software II**

- 실제로 공학 분야에서 부닥치는 응용문제 중 선형 방정식에 기반을 둔 문제를 선정하여 공개 소프트웨어를 사용하여 문제를 풀어봄으로써 문제 해결 능력을 배양한다.

11) **Numerical differentiation and integration**

- 연속 공간에서 이론적으로 정의된 함수의 미분과 적분 기법을 실제로 이산 공간에 기반을 둔 컴퓨터상에서 어떻게 다루는지에 대하여 살펴본다.

- 기본적인 수치 미적분 기법에 대하여 살펴본 후 각 기법의 오차에 대하여 이해를 해본다.

12) **Nonlinear equations and problem solving**

- 이분법과 Newton 방법과 같은 주어진 방정식의 근을 수치적으로 찾아주는 풀이 기법에 대하여 이해한다.

- 실제로 공학 분야에서 부닥치는 응용문제 중 미적분 및 방정식의 풀이에 기반을 둔 문제를 선정하여 풀어봄으로써 문제 해결 능력을 배양한다.

13) **CUDA or OpenCL programming on the GPU**

- GPU의 massively parallel streaming processing 능력을 최대로 활용해주는 병렬 프로그래밍 기법에 대하여 이해한다.

- CUDA 또는 OpenCL API를 사용하여 주어진 예제 문제를 해결해 봄으로서, 병렬 처리를 통한 문제 해결 능력을 배양한다.

14) **Iterative optimization algorithms**

14) **프로그래밍 숙제 예** (다음과 같은 주제 중 선택적으로 연습)

- 다양한 수치 계산 유형에 대한 문제점 분석 및 해결 방안 제시
- 다양한 수치 계산 유형에 대한 코드 최적화 기법 이해
- Intel MMX/SSE를 통한 이미지 처리 기법의 병렬화 구현 기법 연습
- 테이블 기법을 통한 데이터 보간 및 속도 향상 기법 이해
- 3D 렌더링 방법 중의 하나인 Ray Tracing 기법 구현 과정에서의 방정식의 풀이 기법 연습 및 수치 계산의 안정성 이해
- 3D 게임 엔진 중 물리 엔진 개발에 필요한 미분 방정식 풀이 기법 연습
- 3D 애니메이션 기법 중의 하나인 Key Frame 기법을 통한 애니메이션 컨트롤 방법 구현에 필요한 역함수 계산 기법 연습
- 컴퓨터 시스템의 성능 평가 시 해결해야 할 선형 방정식의 풀이 기법 연습
- 웹 상의 방대한 데이터에 대한 데이터 마이닝 시 해결 해야할 선형 대수 계산 연습
- 공학 분야에서 빈번히 발생하는 방대한 크기의 선형 방정식의 풀이 및 안정성 이해

권장 도서 및 문서

- **수치 컴퓨팅**
  - 다음과 같은 전통적인 수치해석 교재
    - R. Burden and J. D. Faires, ***Numerical Analysis*** (8th ed), Thomson Books/Cole, 2005
    - W. Cheney and D. Kincaid, ***Numerical Mathematics and Computing*** (6th ed), Thomson Books/Cole, 2008.
    - 기타
- **GPU 프로그래밍**
  - D. Kaeli et al., ***Heterogeneous Computing with OpenCL 2.0*** (3rd ed.), Morgan Kaufmann, 2015. (참고: 이전 버전도 OK)
  - M. Scarpino, ***OpenCL in Action: How to Accelerate Graphics and Computations***, Manning Publications, 2011.
  - A. Munshi et al., ***OpenCL Programming Guide***, Addison-Wesley Professional, 2011.
  - https://www.khronos.org의 OpenCL 관련 공식문서
  - 기타 GPU 및 GPGPU 관련 자료