A Unified Approach to Formal Description of Ground Motion Prediction Equations

Alexey Gokhberg [1] and Laurentiu Danciu [2]

1 FRAGATA COMPUTER SYSTEMS AG, Switzerland, **2** ETH Zurich, Swiss Seismological Service, Switzerland

Introduction

Algorithms for ground motion prediction equations (GMPE) play a crucial role in implementation of PSHA computations. Hundreds of such algorithms have been developed so far and new algorithms are published every year as result of ongoing research. There is no commonly accepted standard for the formal description of GMPEs and researchers use a broad variety of programming languages for their implementation. As the result, codes for different GMPEs are frequently mutually incompatible and often specific for a certain computing environment.

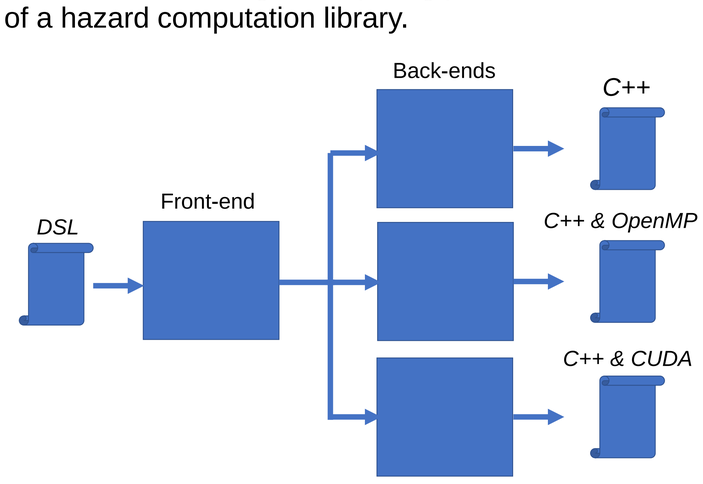
Furthermore, emergence of new high performance computing architectures like GPU accelerators or many-core processors substantially increases implementation complexity of GMPE algorithms and we expect that in the future this complexity will grow with the advent of even more sophisticated computing platforms. Therefore, there exists a clear need for a unified approach towards coding formalized GMPE descriptions, which could be immediately used on a broad range of modern and future computing systems without additional programming effort.

Fortran
Java
Python
MATLAB

MPI
OpenMP
CUDA

Methods

We have designed a domain-specific programming language (DSL) for the formal description of GMPE algorithms. The language defines GMPE logic at a high level of abstraction. Algorithm descriptions are independent of implementation-specific details like architecture of a target computing platform or version of a hazard computation library.



We have implemented a source-to-source compiler that transforms GMPE descriptions into various target programming languages. The compiler consists of a common front-end performing platform-independent analysis of a source code and several back-ends generating code for various target languages. Additional code generators targeting different programming languages and hardware architectures can be implemented if required.

We expect that our approach will allow a researcher who is not an expert in computer science to design GMPEs that can be used on various state-of-the-art high performance computing systems.

Design concepts

A GMPE description contains the following sections:

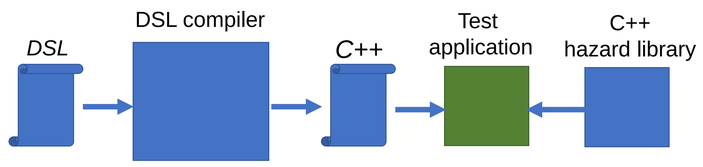
- header;
- list of supported features:
  - tectonic region type;
  - intensity measure types;
  - intensity measure component;
  - standard deviation types;
- list of required context parameters:
  - site parameters;
  - rupture parameters;
  - site-to-rupture distances;
- kernel providing a list of functions;
- constant definitions;
- coefficient tables.

Design of GMPE descriptions has been modelled after OpenQuake. However, unlike OpenQuake, our DSL conceptually describes computation of the mean and standard deviation values for a single site and a single intensity measure type. This approach leads to leaner code and gives the compiler freedom for implementing parallel computations in a platform-specific way.

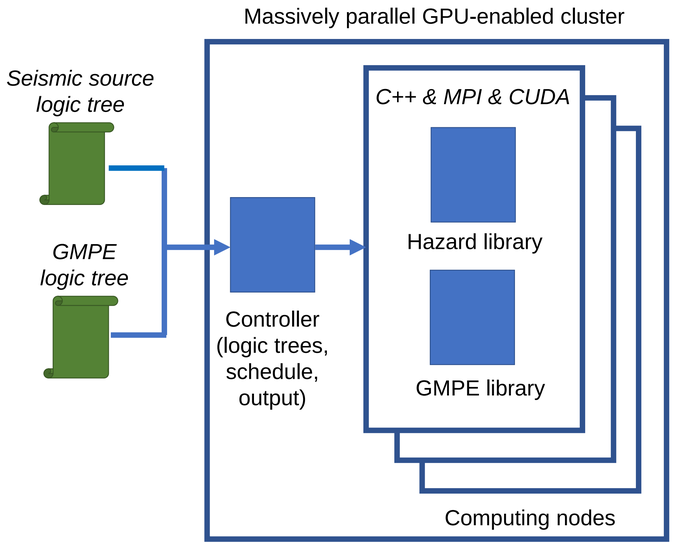
The DSL allows construction of new GMPE descriptions by combining elements of already available descriptions for similar GMPEs thus eliminating redundancy in algorithm codes.

Applications

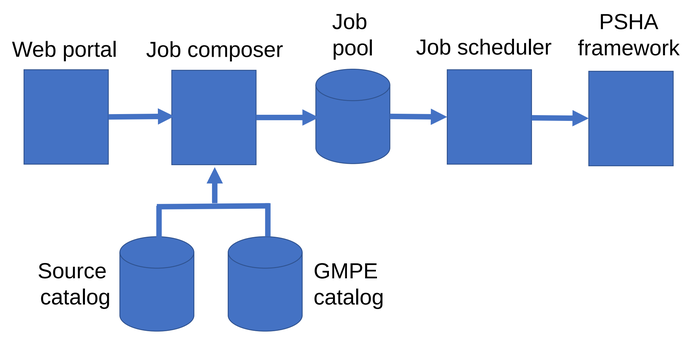
Case 1: Desktop GMPE development toolkit



Case 2: Massively parallel PSHA framework



Future research: High performance PSHA as a service



Results

We have ported around 25% of GMPE descriptions available in OpenQuake 2.1 hazard library from Python to our DSL. All descriptions have been successfully tested against the OpenQuake GMPE testing suite.

We have used our DSL compiler to convert this GMPE library into C++ and CUDA. The generated code has been integrated with the high performance PSHA framework and deployed for evaluation on the GPU-enabled massively parallel computer “Piz Daint” operated by the Swiss National Supercomputing Centre (CSCS). We ran computations for the 2013 European Seismic Hazard Model (ESHM13, Woessner et al 2015); the running time on a single computing node (one NVDIA P100 GPU) is around 7 hours. This test demonstrates suitability of our approach for solving large real-life PSHA problems.

We would like to thank Global Earthquake Model Foundation (https://www.globalquakemodel.org) for developing and publishing OpenQuake, which we used as a source for ideas, algorithms, and test cases. We also thank CSCS for providing “Piz Daint” for the evaluation of our solution.

Example

```
@module{akkar_2014}

@gmpe{AkkarEtAlRjb2014}

@support{TRT} ACTIVE_SHALLOW_CRUST
@support{IMT} PGA PGV SA
@support{IMC} AVERAGE_HORIZONTAL
@support{StdDev} TOTAL INTER_EVENT INTRA_EVENT

@require{site} vs30
@require{rup} rake mag
@require{dist} rjb

@kernel
void main() {
    coeffs C_pga = COEFFS_PGA
    real median_pga =
        exp(compute_mean(C_pga, rup.mag, rup.rake))
    coeffs C = COEFFS
    mean = compute_mean(C, rup.mag, rup.rake) +
        compute_non_linear_term(C, median_pga)
    get_stddevs(C)
    }

real compute_mean(coeffs C, real mag, real rake) {
    real mean =
        C.a1 +
        compute_linear_magnitude_term(C, mag) +
        compute_quadratic_magnitude_term(C, mag) +
        compute_logarithmic_distance_term(C, mag) +
        compute_faulting_style_term(C, rake)
    return mean
    }

void get_stddevs(coeffs C) {
    for_all_stddev_types {
        if (stddev_type == StdDev.TOTAL)
            stddev = sqrt(C.sigma**2+C.tau**2)
        else if (stddev_type == StdDev.INTRA_EVENT)
            stddev = C.sigma
        else if (stddev_type == StdDev.INTER_EVENT)
            stddev = C.tau
        }
    }

real compute_linear_magnitude_term(coeffs C, real mag) {
    if (mag <= CONST.c1)
        return C.a2 * (mag - CONST.c1)
    else
        return C.a7 * (mag - CONST.c1)
    }

real compute_quadratic_magnitude_term(coeffs C, real mag) {...}
real compute_logarithmic_distance_term(coeffs C, real mag) {...}
real compute_faulting_style_term(coeffs C, real rake) {...}
real compute_non_linear_term(coeffs C, real pga_only) {...}

@const{CONST} c1 = 6.75

@coeffs{COEFFS, 5}
    IMT       a1          a2          ...   sigma     tau
    pga       1.85329     0.0029      ...   0.6201    0.3501
    pgv       5.61201     0.0029      ...   0.6014    0.3311
    0.010     1.87032     0.0029      ...   0.6215    0.3526
    0.020     1.95279     0.0029      ...   0.6266    0.3555
    ...
    3.800    -1.39790     0.0029      ...   0.6389    0.415
    4.000    -1.37536     0.0029      ...   0.6196    0.3566

@fetch{COEFFS_PGA, COEFFS, pga}
```

Contacts:
alexey.gokhberg@erdw.ethz.ch
laurentiu.danciu@sed.ethz.ch

FRAGATA

Schweizerischer Erdbebendienst
Service Sismologique Suisse
Servizio Sismico Svizzero
Swiss Seismological Service

ETH zürich