

OpenROAD Runtime Improvisation by updating C code

Sumanto Kar,
Mumbai, India
Jeetsumanto123@gmail.com

Abstract— OpenROAD is a popular Open Source Tool for the RTL to GDS flow. However, it becomes a necessity to improve the tool chain. One such approach would be to optimize the C code written.

Keywords— *OpenROAD Flow Scripts, RTL-to-GDSII flow, open-source tools, automated design, no-human-in-the-loop.*

I. INTRODUCTION (HEADING I)

The digital integrated circuit industry has widely embraced OpenROAD, a streamlined RTL-to-GDSII flow offering a comprehensive design implementation solution. This popularity can be attributed to the efficiency of the design flow and the fact that it is open-source. However, it is necessary to reduce the runtime of OpenROAD by improvising/optimizing the C code scripts. Here we investigate the gpuSolver.cu.

II. IMPROVEMENTS

The before and after implementation of the gpuSolver.cu is illustrated below:

Before:

```
void GpuSolver::cusolvererror(cusolverStatus_t code)
{
    if (code != CUSOLVER_STATUS_SUCCESS) {
        log_>error(GPL,
            1,
            "[CUSOLVER ERROR] {} at line {} in file {} \n",
            cudaGetErrorString(*(cudaError_t*) &code),
            __LINE__,
            __FILE__);
    }
}

GpuSolver::GpuSolver(SMatrix& placeInstForceMatrix,
    Eigen::VectorXf& fixedInstForceVec,
    utl::Logger* logger)
{
    // {cooRowIndex_, cooColIndex_, cooVal_} are the host vectors used to store
    // the sparse format of placeInstForceMatrix.
    nnz_ = placeInstForceMatrix.nonZeros();
    vector<int> cooRowIndex, cooColIndex;
    vector<float> cooVal;
    cooRowIndex.reserve(nnz_);
    cooColIndex.reserve(nnz_);
    cooVal.reserve(nnz_);

    for (int row = 0; row < placeInstForceMatrix.outerSize(); row++) {
        for (typename Eigen::SparseMatrix<float, Eigen::RowMajor>::InnerIterator it(
            placeInstForceMatrix, row);
            it;
            ++it) {
            cooRowIndex.push_back(it.row());
            cooColIndex.push_back(it.col());
            cooVal.push_back(it.value());
        }
    }
}
```

After:

```
void GpuSolver::cudaerror(cudaError_t code)
{
    if (code != cudaSuccess) {
        log_>error(GPL,
            1,
            "[CUDA ERROR] {} at line {} in file {} \n",
            cudaGetErrorString(code),
            __LINE__,
            __FILE__);
    }
}

void GpuSolver::cusparseerror(cusparseStatus_t code)
{
    if (code != CUSPARSE_STATUS_SUCCESS) {
        log_>error(GPL,
            1,
            "[CUSPARSE ERROR] {} at line {} in file {} \n",
            cusparseGetErrorString(code),
            __LINE__,
            __FILE__);
    }
}

void GpuSolver::cusolvererror(cusolverStatus_t code)
{
    if (code != CUSOLVER_STATUS_SUCCESS) {
        log_>error(GPL,
            1,
            "[CUSOLVER ERROR] {} at line {} in file {} \n",
            cudaGetErrorString(*(cudaError_t*) &code),
            __LINE__,
            __FILE__);
    }
}

GpuSolver::GpuSolver(SMatrix& placeInstForceMatrix,
    Eigen::VectorXf& fixedInstForceVec,
    utl::Logger* logger)
{
    // ...
}
```

III. CONCLUSIONS AND RECOMMENDATIONS

In conclusion, the removal of many redundant segments of the code. This would decrease the runtime of the code and hence make the OpenROAD execution faster and more efficient.

REFERENCES

- [1] OpenROAD. [Online]. Available: <https://github.com/The-OpenROAD-Project/OpenROAD-flow-scripts>.
- [2] O. Project. [Online]. Available: <https://github.com/The-OpenROAD-Project/OpenROAD>.
- [3] S. Kar, "https://github.com/Eyantra698Sumanto/OpenROAD-flow-scripts/blob/7nmcontest/7nmDesignContest.md," 2023. [Online].